LINUX • JAVA • VISUAL BASIC • DEMOSCENE • VISUAL C++ • PROGRAMACIÓN 3D • SQL

PROGRAMACIÓN ACTUAL

AÑO 2 • NÚMERO 13

ARGENTINA 10$ • CHILE 3000$ • PORTUGAL 1250 ESC (CONT)

POWER J 2.0

LA SOLUCIÓN PARA LAS EMPRESAS

SÓLO 995 ptas

SECCIÓN NIVEL BÁSICO

386 en **ENSAMBLADOR**, Clases en **PROGRAMACIÓN C++**, Grafos en **CURSO DE PROGRAMACIÓN**

SECCIÓN BAJO NIVEL

Problemas con mapeado de texturas en **ENTORNOS 3D**, Modos **SVGA** en **PROGRAMACIÓN GRÁFICA**, Texto en modo gráfico **DEMOSCENE**

SECCIÓN LAS EMPRESAS DEMANDAN

POO en **VISUAL BASIC 5.0**, ActiveX en **VISUAL C++ 5.0**, Las DBtools.h en **BASES DE DATOS**, **SET** en **COMERCIO ELECTRÓNICO**, Configuración en **WINDOWS NT**

SECCIÓN CAMPUS ACTUAL

Control de versiones en **LINUX ACTUAL**, Utilidades **TCP/IP** en **REDES LOCALES**, Modelo de objetos en Javascript en **WWW**

ACTIVEX Y VISUAL C++

EN EL CD ROM

SOFTWARE GNU

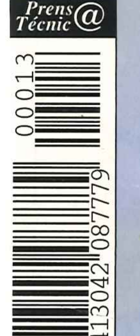
- Compilador GNU C v2.7.2.1 para DOS.
- Compilador GNU C v2.80 para Windows 95.
- Compilador GNU GCC v2.80 para Unix.

DEMOS

- Norton Antivirus para Windows 95, NT 4.0
- Norton CrashGuard.
- Visual Cafe 1.0.
- DIV GAMES STUDIO

Y UTILIDADES DE PROGRAMACIÓN

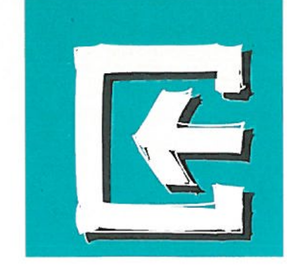
• ActiveFile 1.2 • AddSoft 1.12 • CGI Expert 3.03b • CodeSMART 2.2 • CooScale 1.1 • eAuthor Help 2.05 preview 5 • EZ Macros 3.0 • Form To VB Export Wizard 1.3 • HexDecCharEditor 1.02 • Lf Directory Tree 1.2 • MultiLanguage Pack 2.01 • PiXCL Tools 4.1 • Project Analyzer 4.1.05 • Setup Specialist 97b • Stub Generator 1.1 • XShell 1.0 • y muchas más.



MODELO DE OBJETOS EN JAVASCRIPT AVANZADO







Edita PRENSA TÉCNICA S.L.
pactual@prensatecnica.com

Director/Editor
Mario Luis

Redactor Jefe
Eduardo Toribio

Edición
Charo Sánchez

Colaboradores
María J. Recio, Teresa Madrid, Jorge R. Regidor, Carlos Falcato, Juan Manuel Martín, Luis Martín, Fernando J. Echevarrieta, Santiago Romero, Carlos Poyatos, Antonio Ruiz, César Sánchez, Pedro Antón, Felipe Bertrand, Enrique de Alarcón, Fernando de la Villa, José Remiro, José M. Jiménez, Ramón García

Jefa Maquetación
Carmen Cañas

Maquetación
Pedro Bustos,
Marga Vaquero,
Manuel J. Montes

Portada
Carlos Sánchez

Publicidad
Marisa Fernández

Suscripciones
Sonia Glez. Villamil

Filmación
Grafroprint

Impresión
Gráficas 82

Duplicación del CD-ROM
M.P.O.

Distribución
SGEL

Redacción, Publicidad y Administración
C/ Alfonso Gómez 42, nave 1-1-2
28037, MADRID, ESPAÑA
Telf.: (91) 304 06 22
Fax: (91) 304 17 97

Depósito legal: M-8914-1997

ISSN: 1137-5531

AÑO 2 • NÚMERO 13
Copyright 30-6-98

PRINTED IN SPAIN

Programadores y Televisión

Bienvenidos a un nuevo número de Programación Actual.
Hace poco comentaba con gente relacionada con nuestro sector el poco tiempo que dedicamos los programadores a la televisión, y, sorprendentemente, era un denominador común entre todos los reunidos. No sé si queda mal decir que uno ve mucha televisión o es que, realmente, nuestro tiempo de ocio lo dedicamos al ordenador y poco más. Tal vez, lo cierto es que a muchos de los programadores no les interesa la mayor parte de la programación televisiva, y puede que les gustara mucho más tener su propio canal de televisión. Y digo esto porque, aunque suena a descabellado, no lo es tanto hoy en día; de hecho, esta conversación tenía lugar a propósito de la invasión de canales temáticos en las nuevas televisiones digitales. Comentábamos que el canal C:, de Canal Satélite Digital, podría resultar muy beneficioso, con un gran potencial, aunque ahora sea minoritario, y que sería aún más provechoso para nosotros tener un canal temático propio dedicado al mundo de la programación. Todos puestos de acuerdo en lo positivo de la idea, incluso llegamos a confeccionar una parrilla de programación completa, que debía ser educativa, didáctica y, sobre todo, amena y divertida. Sería algo así como una revista, como esta revista, pero con el poder infinito del medio audiovisual. Debía estar también orientada a todo tipo de programadores, con novedades de productos, concursos, bolsa de trabajo y un poco de todo. Por eso, desde esta tribuna informativa, lanzamos la idea esperando que caiga en alguna mesa de algún directivo, pues consideramos de verdad que existe una audiencia potencial a tener en cuenta, y que sólo por el hecho de que la programación es una tecnología con gran demanda en el sector laboral habría mucha gente interesada en sacar partido de los distintos cursos. En fin, si hay canales específicos de moda o de pesca por qué no uno en exclusiva de programación.

Y por lo que respecta a este número 13, tocaremos madera, comentar que nuestro tema de portada se lo hemos dedicado a PowerJ, herramienta de programación de applets y aplicaciones Java de la filial canadiense de Sybase, Powersoft, hemos incluido un análisis del producto en el interior de la revista. Otras secciones que destacamos son Visual C, dedicada a la creación de controles ActiveX, y Programación en Internet, con su segunda entrega de Javascript avanzado. El resto de las secciones continúan su progresivo avance esperando que cumplan su propósito, que no es otro que desarrollar y completar la formación del lector. Por supuesto que, además de la sección nueva dedicada a Windows NT, habrá otras novedades próximamente. La primera ya está en el presente número en forma de correo del lector para temas GNU/Linux, pues eran muchas las dudas que llegaban desde este frente, lo que nos ha animado a dedicar dos páginas a este tema.

En cuanto al Cd-rom, este mes aparecen bastantes megas de software gnu, el compilador gnu de C++ para Windows, diversas herramientas, etc. Aparte, volvemos a incluir la herramienta de desarrollo de videojuegos DIV y adjuntamos, como ya habréis visto, un pequeño manual que nos demandaban bastantes lectores desde hace ya un par de meses. Bueno, creo que ahora ya nadie tiene excusa para no realizar sus primeros pinitos.

Un cordial saludo,

Eduardo Toribio
etori@ergos.es

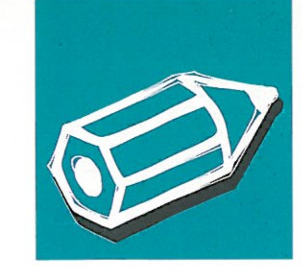


SUMARIO

En Portada

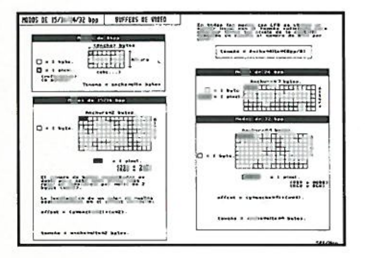
En este número 13 viene a nuestra portada el completo entorno de desarrollo de programas Java Powerj de la canadiense desarrolladora Powersoft. Realizamos un cover a esta herramienta que permite tanto la creación de pequeños applets como de grandes aplicaciones que hagan un uso intensivo de bases de datos.

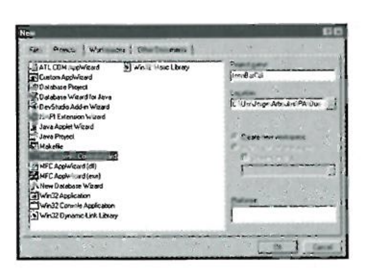




Sumario

LAS EMPRESAS DEMANDAN





CAMPUS ACTUAL

En el cd-rom de este mes se encuentra además de los fuentes y utilidades relacionadas con los distintos cursos, gran cantidad de software gnu, utilidades share de programación, los últimos navegadores para la red y las demos operativas de DIV GAMES STUDIO y VISUAL CAFE de Symantec.

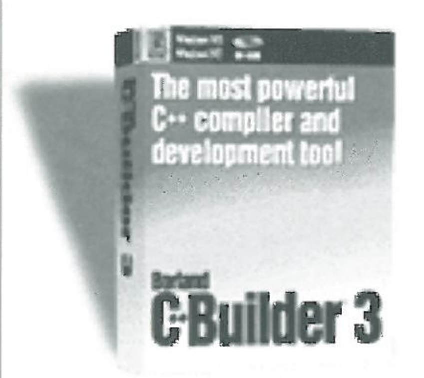
DEMOS OPERATIVAS
Norton Antivirus para Windows 95, NT 4.0 y 3.51.
Norton CrashGuard.
Visual Cafe 1.0.
DIV GAMES STUDIO, toda la información en el manual de regalo.
GARGOSCENE
Número 1 de la revista electrónica Gargoscene desarrollada por miembros de la asociación de desarrolladores STRATOS.
SOFTWARE GNU
En \GNU el siguiente software distribuible bajo licencia GNU:
Compilador GNU C++ v2.7.2.1 para DOS.
Compilador GNU C++ v2.80 para Windows 95.
Compilador GNU GCC v2.80 para Unix.
PROGRAMACION
En el directorio \UTILS\PROG recopilación de las últimas novedades de software share de programación. Destacamos aquí a modo de introducción:
ActiveFile 1.2
ActiveFile es un control ActiveX que está diseñado como un control para manejar ficheros y directorios desde servidores Web activos. Requeriere ActiveX.
AddSoft 1.12
AddSoft ahorra tiempo al que lo utiliza, ya que busca shareware por toda la Red con ayuda de los buscadores más conocidos.
CGI Expert 3.03b
CGI, Win-CGI, ISAPI y NSAPI son interfaces que permiten crear documentos HTML dinamicos e interactivos.
CodeSMART 2.2
Explorador de proyectos que procesa código y crea accesos directos a rutinas, declaraciones, constantes, tipos, etc.
CooScale 1.1
CooScale una clase de Java que permite generar una barra de progresión. En ella se pueden incluir sonidos y animaciones.
eAuthor Help 2.05 preview 5
Este poderoso programa aporta todo lo necesario para escribir ficheros de ayuda en HTML que cumplan las especificaciones estándar.
EZ Macros 3.0
EZ Macros es un grabador de macros que soporta cientos de combinaciones de teclas "calientes" y el que se puede utilizar tanto el teclado como el ratón. Las macros pueden ser grabadas o escritas manualmente.
Setup Specialist 97b
Setup Specialist es una herramienta visual para crear instalaciones de programas. Puede generarlas para plataformas de 16 y 32 bits.
HexDecCharEditor 1.02
HexDecCharEditor, como su largo nombre índica, es un editor decimal/hexadecimal que incluye búsquedas, reemplazamientos, marcas, y todo lo necesario para hacer de este editor una herramienta imprescindible.
Lt Directory Tree 1.2
LT Directory Tree es un control ActiveX para Visual Basic 4 y 5 que permite dar protección a unidades de disco duro o carpetas. Además da acceso directo al panel de control y permite mover directorios completos.
MultiLanguage Pack 2.01
MultiLanguage Pack es un componente para Delphi que hace que las aplicaciones puedan ser localizadas por el autor. Muy útil para crear actualizaciones o para desintalar versiones anteriores en caso de instalar una nueva.
PiXCL Tools 4.1
PiXCL Tools permite desarrollar aplicaciones gráficas aportando una gran variedad de comandos para crear y mostrar imágenes, sonidos y texto. Este lenguaje interpretado ahorra el aprendizaje del código del interfaz gráfico de Windows.
Más información en las páginas dedicadas al contenido del cd-rom.



NOTICIAS

BORLAND PRESENTA C++ BUILDER 3

El Entorno de desarrolo y compilador C++ avanzado



Una solución corporativa que le ofrece el entorno de desarrollo y compilación C++ más potente, rico en prestaciones y completamente integrado jamás creado por Borland. C++Builder 3 le permite construir aplicaciones de gestión empresarial con herramientas de Bases de Datos integradas. Ofrecer aplicaciones Web que gestionen Bases de Datos y convertir los datos de empresa o clientes en información de alto valor añadido. Todo esto completando proyectos C++ en tiempo record y sin sobrepasar estimaciones de tiempo y presupuesto.

C++Builder 3 aumenta la productividad en C++ gracias a su entorno de desarrollo visual de alto rendimiento.

• Desarrollo ActiveX/ATL mediante asistentes en un solo paso.
C++Builder 3 crea fácilmente controles ActiveX de alto rendimiento válidos para C++, Visual C++, Java™, Visual Basic y PowerBuilder.

• Administración de proyectos para gestionar código complejo y compilar varios ejecutables de destino, como EXE, LIB y DLL.

• Permite abrir proyectos de C++, enlazar su propio código OWL y MFC, y convierta los archivos RC de Resource workshop.

• Desarrollo de aplicaciones de bases de datos sofisticadas, gracias a los más de 25 componentes enlazados con datos. El proceso se simplifica con el Database Explorer, que permite editar metadatos, y el Data Dictionary, que garantiza la integridad de los datos..

• Compatibilidad con estándares: C++, Windows 95/NT, Win32 API, procesos multihilo, ActiveX/ATL, COM, STL, MFC 4.2 y OWL.

• Asistentes ActiveForm— Publicación de datos en la Web, en un solo paso.

• Desarrollo rápido de aplicaciones ANSI C++. Codificación visual y tradicional con más de 130 componentes reutilizables, incluidos menús, cuadros de diálogo, visualizadores de datos, controles para Windows 95 y NT, asistentes y otros.



BORLAND LLEVA EL DESARROLLO JAVA AL AS/400

Con cada copia nueva de OS/400 se incluirá una versión de prueba de JBuilder/400

Borland e IBM han anunciado el inicio de relaciones comerciales tomando como base la nueva herramienta de desarrollo Java de Borland para la plataforma AS/400, la suite cliente/servidor JBuilder/400. Dicho anuncio se llevó a cabo al tiempo que se realizaba el lanzamiento, por parte de IBM, de la nueva versión del sistema operativo AS/400, OS/400 V4R2 y su máquina virtual Java. Como parte del acuerdo, en cada AS/400 y en cada actualización del sistema operativo lanzada por IBM, se incluirá una versión de prueba, con una duración de 90 horas, de JBuilder/400. Los clientes más cualificados podrán recibir una versión de prueba de dicha suite cliente/servidor registrándose en la página web de Borland/400 en la dirección *http://www.borland.es/borland400/*. Borland JBuilder es una familia de herramientas de desarrollo visual para crear aplicaciones de alto rendimiento e independientes de la plataforma que utilizan el lenguaje de programación Java. El entorno dimensionable basado en componentes de JBuilder está diseñado para todos los niveles de proyectos de desarrollo en redes de información, que van desde applets y aplicaciones hasta sistemas multiniveles distribuidos de cliente/servidor y empresa.

Asimismo, Borland e IBM han estado trabajando en un esfuerzo conjunto de I+D para mejorar el desarrollo de Java empresa mediante la integración de JBuilder con el proyecto San Francisco de IBM. Como parte de este esfuerzo, ambas empresas han estado creando herramientas visuales, asistentes y componentes orientados a objetos para optimizar el aprendizaje y la productividad en lo referente al proyecto San Francisco.

En cuanto a la familia Borland de herramientas de desarrollo AS/400 interoperables, Delphi/400, C++Builder/400 y JBuilder/400, mejoran la inversión de los clientes en tecnología IBM AS/400, satisfaciendo las necesidades de desarrollo cliente/servidor rápido de la comunidad AS/400.

Por otra parte, Hispanian Tecnologic, centro especializado en formación informática avanzada, ha presentado un curso gratuito de Visual J++, con una totalidad de seis entregas mensuales, a partir del presente mes de marzo, lo que permitirá a los alumnos conocer la plataforma de desarrollo más utilizada en Java. Para ello, las personas interesadas sólo tendrán que consultar la página web de dicha agrupación en la dirección *www.hispan.com.* El único requisito para realizar este curso es registrarse, además, Hispanic Tecnologic sorteará, entre todas las personas registradas, un curso a distancia de Visual J++ con el que el alumno podrá profundizar en el tema y obtener, si así lo desea, el certificado correspondiente. La primera entrega consta de un par de sencillos proyectos para presentar este entorno, demostrando que Visual J++ consiste en un entorno de desarrollo de propósito general con Java, que permite crear applets y aplicaciones independientes de la plataforma. El hecho de que se haya elegido este tipo de curso es porque Visual J++ consiste en la plataforma de desarrollo que Microsoft propone para la creación de aplicaciones en Java., mientras que éste es un lenguaje de programación orientado a objetos que, en buena medida proviene de C y C++, y, por tanto, hereda la mayoría de su sintaxis y mecanismos de orientación a objetos.

SUN PRESENTA EL PRIMER SERVIDOR DE ALMACENAMIENTO INTELIGENTE

Este servidor surge como continuación de la redefinición de soluciones de la compañía

Se trata del Sun StorEdge A7000 Intelligent Storage Server, que cuenta con el software StorEdge DataShare y ofrece a los clientes una sólida compartición de información entre sistemas mainframe, UNIX y NT, además de ser soportado por plataformas heterogéneas, incluido el entorno operativo Solaris. Por otro lado, también se ha anunciado el software Sun StorEdge DataShare, el único software del sector que permite la compartición directa de datos sin necesidad de realizar transferencia de ficheros. El A7000 y DataShare son piedras angulares en la implementación de la arquitectura Intelligent Storage Network de Sun. A diferencia de otros productos, el sistema Sun StorEdge A7000 alberga dos servidores UNIX con cuatro procesadores, lo que permite a los clientes implementar aplicaciones de almacenamiento, tales como la compartición de información, copia de datos remotos y copias de seguridad, con la consecuente disminución de la carga del ordenador central. Esta compartición directa de la información reduce los ciclos de proceso, lo que aligera la carga tanto en mainframes como en sistemas abiertos, simplifica la gestión de la infraestructura y reduce los costes de explotación. La propiedad de esta inteligencia incorporada es que ofrece a los usuarios un alto nivel de fiabilidad, disponibilidad y mantenimiento (RAS) mediante la combinación de los niveles RAID 1, 0+1 y 5, y el diagnóstico telefónico y remoto; asimismo, incorpora memoria caché NVRAM duplicada que garantiza a los clientes la disponibilidad permanente de los datos de la memoria caché, incluso en el caso de que se pierda o se deteriore una copia.

Por otra parte, Sun ha lanzado **Java Supplement** para su software Solstice Enterprise Manager, una solución multiprotocolo para gestión de redes destinada a empresas de telecomunicaciones, proveedores de servicios Internet (ISP) y grandes corporaciones. Este novedoso software ofrece un conjunto de APIs de Java para el desarrollo de aplicaciones personalizadas de gestión de redes, así como Java Managemet Console, aplicación basada en exploradores que permite a los administradores controlar la red desde cualquier lugar y en cualquier momento en la nueva red orientada a servicios. Con Solstice Enterprise Manager 2.1 Java Supplement, los administradores de redes que reciben llamadas durante las 24 horas del día pueden responder de una forma rápida a las alarmas de la red sin salir de casa; asimismo, los proveedores de servicios pueden facilitar a sus clientes una visión segura y en tiempo real de las redes gestionadas, y, finalmente, el personal de operaciones tiene la posibilidad de facilitar a la dirección, de un modo más exacto, informes actualizados sobre el estado real de los servicios críticos de la red. Otro tipo de ventajas y funcionalidades que ofrece el nuevo Solstice son APIs Java de alto nivel para la manipulación de objetos gestionados, soporta arquitecturas de clientes ligeros que toman la información de servidores específicos a través de un adaptador de gestión Java que posibilita la distribución del proceso, y, para finalizar, posee una consola de gestión con un alto grado de personalización que permite a los operadores y administradores de red consultar topologías de red, visualizar datos de alarmas y recopilar datos sobre dispositivos, todo ello, desde una aplicación Java de clientes ligeros y de fácil manejo. El producto estará disponible en el mercado en el primer trimestre del presente año.

Finalmente, otro de los productos revolucionarios de la empresa americana ha sido el anuncio de lanzamiento del primer conjunto de productos de gestión de redes basados en web, diseñado de forma específica para la creación de redes orientadas a servicios. Las nuevas redes Orientadas a Servicios (Service-Driven Networks) representan las novedosas redes multifabricante con capacidad de respuesta, modulares, fiables y dinámicas que los proveedores de telecomunicaciones y grandes empresas corporativas necesitan implementar con el fin de potenciar sus empresas y prestar un mejor servicio a sus clientes o usuarios. Dichas redes, basadas en una nueva generación de tecnología de gestión, ofrecen la posibilidad de elección entre una serie de innovadores servicios, por ejemplo, itinerancia unificada, difusión de noticias, compras en línea, etc., de una manera más rápida y a un coste más bajo que nunca.

Para más información:
http://www.sun.com

PROGRAMACION EN COBOL VISUAL PARA WINDOWS DE NIGSUN

Se trata de un producto orientado a programadores de Cobol bajo el sistema de Windows



La compañía NIGSUN ha presentado Objective COBOL (OC), un producto que, entre otras características, soporta una jerarquía de clases llamada VCL (Visual Component Library) que consta de un conjunto extenso de controles visuales y no visuales, los cuales realizan acciones determinadas por un conjunto de propiedades, métodos y eventos que son programables tanto en tiempo de diseño como en tiempo de ejecución. Por otro lado, un objeto puede contener múltiples propiedades y eventos, todos ellos accesibles desde el lenguaje; el acceso a estos elementos se realiza mediante la sintaxis estándar de la programación a objetos. Otra característica es que los eventos de un objeto marcan la secuencia de ejecución de un programa y es en dichos eventos en donde se codifica el Cobol orientado a objetos; asimismo, Objective COBOL soporta expresiones numéricas, alfanuméricas o lógicas en cualquier instrucción, es decir, en OC se puede implementar cualquier tipo de expresión dentro de una instrucción aumentando así el rendimiento y la productividad. Se añaden nuevas extensiones al lenguaje para soportar el uso de subrutinas con paso de parámetros y, por lo tanto, la posibilidad de declarar variables locales; finalmente, OC ha introducido una novedosa sección, denominada Define Section, que está pensada para que el cliente o usuario escriba menos código en sus programas fuentes. Básicamente, lo que hace es sustituir todo el código que el usuario escriba en dicha sección por un Identificador que, posteriormente, será empleado a partir de la Procedure División, siendo ideal para generar mantenimientos estándares.

Para más información:
http://www.nigsun.es



Informa: Ramón García

Noticias linux • Noticias linux • Noticias linux

DEBIAN 2.0 YA ESTÁ EN EL MERCADO

Debian, al igual que Linux, es una distribución única desarrollada por un grupo de voluntarios en Internet, gracias a lo cual tiene la mayor colección de programas. Actualmente hay 1439 paquetes en la distribución principal, que se pueden usar, copiar o modificar incluso para fines comerciales, 79 en contrib, 222 en non-free (programas que no son libremente distribuibles), y 21 en non-US (programas relacionados con criptografía, y que por eso no se pueden exportar de Estados Unidos). Esta división se debe a que los desarrolladores de Debian se encuentran entre los defensores más fuertes del software de libre distribución.

Debian 2.0 incluye las últimas versiones estables de los componentes fundamentales del sistema operativo, tales como:

- Kernel 2.0.33 que, seguramente, incluirá soporte FAT32 y nombres largos Joliet en los CDs.
- Librería de C estándar de GNU 2.0.7 (o Linux libc6).
- GCC 2.7.2 y EGCS 1.0.1, ya que GCC 2.8.0 tiene errores importantes y GCC 2.8.1 acaba de salir y no da tiempo.
- XFree86 3.3.2 que lo más probable es que incluya Netscape.

Aparte de otras características nuevas:

- No será necesario decidir qué paquetes se quieren instalar. El usuario puede elegir entre varias opciones, tales como "sistema básico", "sistema básico con X", etc.
- El interfaz de usuario de la instalación usa la librería newt creada por Redhat Software.
- Compatibilidad con aplicaciones para versiones anteriores de la librería C. Las librerías dinámicas de Debian están enlazadas de tal forma que el editor de enlace dinámico puede distinguir entre las que dependen de GNU libc 2 o las que necesitan la librería anterior.
- Menú en X. Todas las aplicaciones se añaden al menú de X al instalarse.
- Control de calidad. La distribución Debian ha desarrollado tres mecanismos de control de calidad, mientras que Debian tiene un mecanismo sofisticado para gestionar los errores encontrados. Si un desarrollador no responde a los errores más graves puede ser expulsado.

Por otro lado, destacar que Debian ha desarrollado un programa, llamado deblint, que detecta los errores más frecuentes de los programas, y puede probarse durante un mes para eliminar todas las errores que puedan quedar. Asimismo, funcionará tanto en PCs como en Amigas, si bien no está claro si logrará funcionar en Sparc, y es muy poco probable que dé tiempo a que funcione sobre Alpha y Power PC.

En el futuro, Debian incluirá un programa fácil de usar para instalar y desinstalar componentes, pero, desgraciadamente, no estará disponible para Debian 2.0. Se llama Deity y puede verse en la dirección *www.debian.org/~jgg/deity/*. Además, incluirá el sistema de administración COAS, desarrollado por Caldera.

La gran ventaja de Debian respecto a otras distribuciones es la colección de programas. Por ejemplo, Debian incluye programas como lftp (un sustituto de FTP, más fácil de usar y más potente), librerías de programación CORBA, el depurador de programas DDD, un emulador de HP48, programas de cálculo numérico como Octave y Yorik, etcétera.

La mayoría de estos programas no están incluidos en ninguna otra distribución de Linux. Para finalizar comentar que Debian es una distribución aconsejable para usuarios con experiencia.

CALDERA OPEN LINUX 1.2. EL NUEVO DISTRIBUIDOR LINUX DE CALDERA

Caldera acaba de publicar una nueva versión de su distribución de Linux, que se puede bajar gratuitamente por Internet, la versión Lite, en *ftp.caldera.com*.

Caldera OpenLinux consiste en una distribución de Linux fácil de instalar, que soporta detección automática de dispositivos (aunque no funciona con muchos de ellos como, por ejemplo, tarjetas de red PCI). Además, el programa de instalación está en varios idiomas, entre ellos el castellano. sin embargo, el interfaz de usuario de configuración e instalación (LISA) es poco avanzado, pues aunque está basado en el tradicional "dialog", pega pantallazos poco agradables.

Caldera OpenLinux 1.2 incluye la versión 2.0.33 del núcleo, la 5.4.38 de la librería C y XFree86 3.3.1, si bien aún no se ha pasado a GNU libc 2, como ya lo han hecho Redhat y Debian. Asimismo, incluye la colección típica de software, compiladores, servidores de red, habitual en las distribuciones Linux. Esta versión quizá destaque sobre las anteriores por incluir versiones actualizadas de los programas. Una característica llamativa es que no incluye el editor GNU Emacs; en su lugar incluye Xemacs.

Este novedoso distribuidor de Caldera está disponible en tres versiones: lite, gratuita disponible por Internet; base, de bajo precio; y estándar, más cara. La primera, lite, tiene una versión de evaluación del escritorio "looking glass", un administrador de aplicaciones similar al de Windows 3.1. La segunda, base, incluye ese programa completamente licenciado y la suite de office StarOffice 4.0 para uso no comercial, y el servidor X de Metro, mientras que la versión estándar incluye StarOffice 4.0 con permiso para uso comercial, el servidor de HTTP Netscape FastTrack, DR-DOS, la base de datos Adabas y herramientas de administración de redes Novell.

Además, Caldera venderá un servidor Novell completo NDS, por un precio inferior a los 4.000 $. El producto incluirá ordenador, sistema operativo y programa servidor NDS.

Para más información:
http://www.caldera.com

OTRAS DISTRIBUCIONES

Otras distribuciones menos conocidas que han publicado nuevas versiones son:

- TurboLinux, una distribución comercial de Linux con paquetes RPM (http://www.turbolinux.com). Acaban de publicar la versión 1.2.
- Stampede, otra distribución de Linux creada por voluntarios. Utiliza un formato de paquetes propio. Más información en su página: *http://www.stampede.org*



Noticias linux • Noticias linux • Noticias linux

HERRAMIENTAS DE PROGRAMACION

GCC 2.8.1
Ha salido una nueva versión del compilador de C de GNU. Esta versión corrige los problemas encontrados en GCC 2.8.0. Por tanto, es muy aconsejable que se actualicen los que hayan decidido usar GCC 2.8.0. La actualización no añade nuevas funciones.
GCC fue uno de los primeros programas del proyecto GNU y uno de los primeros compiladores optimizantes de calidad. Actualmente está un poco atrasado. En C++ genera código lento, debido a su incapacidad de eliminar en las optimizaciones las estructuras (ver http://www.cygnus.com/ml/egcs/1998-Mar/0058.html). Quizá parte del problema se deba al modelo de desarrollo cerrado tradicional del proyecto GNU, al contrario que el desarrollo de Linux. Un grupo de especialistas ha creado un grupo de trabajo, cuyo objetivo es desarrollar características nuevas y experimentales de GCC, pero de forma abierta. El proyecto se llama EGCS. En http://egcs.cygnus.com se puede obtener más información. Debian GNU/Linux ya incluye el compilador experimental EGCS.

C-Forge - Entorno de programación integrado comercial para Linux
Ha sido publicado "Code Forge", que es un entorno de programación integrado para Linux. Para aquellos que estén acostumbrados a usar entornos integrados en Windows puede ser una buena solución. Tiene un interfaz gráfico basado en Motif. Desde Code Forge se puede editar, compilar y depurar programas. Por Internet se puede obtener y comprar una versión de demostración.
http://www.codeforge.com

Source Navigator 4.0 - Entorno de programación comercial con referencias cruzadas
Cygnus Solutions, la empresa que da soporte al software de GNU en sistemas empotrados, ha desarrollado Source Navigator. Quizá lo que distingue a Source Navigator son las referencias cruzadas. Por ejemplo, si se pincha en una llamada a una función, podemos saltar al sitio donde se define la función. Lo mismo sucede con estructuras o variables externas.
Source Navigator puede entender C, C++, Java y Fortran. En la versión pagada, puede añadírsele cualquier otro lenguaje.
Source Navigator puede usar métodos de control de versiones CVS ó RCS. Es posible que su principal defecto sea el interfaz de usuario.
http://www.cygnus.com

APLICACIONES

XFree86 3.3.2
Ha salido una nueva versión del sistema de ventanas X. Las mejoras que incluye son:

- Se han solucionado todos los problemas de seguridad conocidos. En sistemas multiusuario, la actualización es muy aconsejable.
- Funciona con más tarjetas de vídeo: chipset ET6100. Matrox Millenium II AGP, chipsets de Trident Cyber9397, 3DImage975, 3DImage985 y TGUI9685, Number Nine Imagine 128 Revolution, S3 Virge MX, S3 Virge GX2, Matrox Mistique y Cirrus CLGD755x.
- Mejoras en los drivers de algunas tarjetas. Nuevas aceleraciones en W32, chipsets Tseng, chipsets NV1, Riva128, todos los chipsets TGUI. Además, la arquitectura de aceleración de XFree86 (XAA) puede dibujar de forma acelerada líneas discontinuas y polígonos llenos mediante llenado de trapezoides.

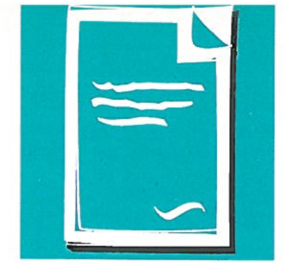
XFree86 se puede obtener por Internet de
http://www.xfree86.org

El código fuente de Netscape será de libre distribución
Netscape ha publicado la licencia de software del código fuente de su navegador. La licencia ha sido diseñada de forma que protege los derechos de Netscape a crear productos comerciales, pero permite la distribución comercial y la modificación. Por lo tanto, cumple con las directrices de Debian de software libremente distribuible (Debian Free Software Guidelines), criterio habitual para decidir si un programa es libre. La nueva licencia se llama NPL (Netscape Public License). Es similar a la licencia GPL de GNU. Prohibe la creación de trabajos derivados que no sean de libre distribución. Esto significa que pronto todas las distribuciones de Linux incluirán Netscape. Bruce Perens, presidente de Software in the Public Interest, ha expresado su alegría, aunque ha dicho que la licencia tiene algunos pequeños defectos que deben ser solucionados.
A finales de marzo será publicado el código fuente de Netscape. El código fuente será completo, excepto aquellas partes que Netscape no puede distribuir, tales como Java o el soporte de encriptación. Por tanto, las sospechas del proyecto GNU sobre si Netscape.
http://www.mozilla.org/NPL

Lyx 0.12.0: interfaz fácil de usar para LaTeX
Lyx es un procesador de textos visual que genera salida en LaTeX. Acaba de salir la versión 0.12..0. Es completo: incluso se pueden crear de forma visual tablas y ecuaciones. Se pueden crear documentos bastante sofisticados. Sin embargo, tiene algunos errores e incomodidades. Su principal limitación es que no puede importar documentos LaTeX ya existentes. Sin embargo, sus creadores han anunciado que una versión futura podrá hacer esto.
Lyx usa la librería XForms. Se puede distribuir libremente según las condiciones de la licencia GPL.
Actualmente estamos viendo algunos procesadores de textos nuevos, como Maxwell (http://www.eeyore-mule. demon.co.uk/) y Papyrus.
http://la1ad.uio.no/lyx/about.htm

Nueva versión BETA (3.21.26-gamma) de la base de datos MySQL
TCX ha publicado una nueva beta del servidor de bases de datos MySQL. MySQL es un gestor de bases de datos muy rápido, adecuado para consultas desde páginas de HTML.
MySQL se puede usar desde clientes Windows porque incluye controlador de ODBC.
La licencia de MySQL permite usarla libremente, pero no se puede distribuir con fines comerciales. Por ello, las distribuciones de Linux no la pueden incluir. De ahí que la base de datos más popular siga siendo PostgresQL.
La versión definitiva saldrá probablemente dentro de dos meses.
http://www.mysql.net/

Fortify: encriptación potente para Netscape
Fortify es un parche para el ejecutable de Netscape que añade encriptación potente de 128 bits. Se puede bajar de http://www.fortify.net/.



Autor: Ramón García

Netscape, el último rival de Microsoft

Las dos últimas décadas de la Historia de la Informática personal se han caracterizado por el monopolio de Microsoft. Gracias a su habilidad haciendo programas fáciles de usar (pero no estables) y a sus métodos dudosos ha logrado derribar, hasta el momento, a todos sus competidores. Netscape ha sido el último episodio de esta historia.

Tras el acuerdo entre el Departamento de Justicia de Estados Unidos y Microsoft, Novell decidió retirar el sistema Novell DOS. Microsoft se quedó sin competencia importante en sistemas operativos. Y entonces llegó Windows 95, y, tras éste, llegó Internet. Pero también llegó Netscape. Y con Internet vinieron los primeros fracasos de Microsoft, ya que Microsoft Network consiguió muy pocos clientes, aparte de que Netscape se benefició de llevar más tiempo en el negocio de Internet, con lo que creció rápidamente, y tanto su navegador como su servidor triunfaron. De hecho, los mayores bancos españoles usan el servidor Netscape: Banesto, BBV, Banco Central Hispano, Banco de Santander y Bankinter. La única excepción es Argentaria, que usa el servidor de Microsoft.

Pero a Microsoft no le gusta tener competencia. Así que utilizó sus estrategias habituales: aprovechó al máximo su control del sistema operativo y mejoró todo lo posible la facilidad de uso. Para asegurarse el éxito decidió regalar Internet Explorer. Y obligó a los vendedores de ordenadores que incluyen Windows 95 OSR2 preinstalado a incluir también Internet Explorer, lo que provocó nuevos problemas con el Departamento de Justicia. Actualmente, usan Internet Explorer un 70 % de los usuarios, lo que demuestra el éxito que ha tenido Microsoft. Algo similar hizo con Windows NT. Publicó dos ediciones: Workstation y Server. La primera tiene una licencia que prohíbe usarlo con más de diez clientes, mientras que la edición Server incluye un servidor de HTTP. Para O'Reilly & Associates, que como Netscape vende un servidor para Windows NT, lo que hace Microsoft es cobrarnos por su servidor aunque no lo queramos, de forma que es prácticamente inviable comprar un servidor alternativo. Según la consultora Netcraft, el servidor de HTTP más popular es Apache (un servidor de libre distribución para Unix), seguido de los servidores de Microsoft (21,8 % incluyendo Internet Information Server y otros) y Netscape (9,98 % entre todos sus servidores).

Regalar un programa como Internet Explorer, en el que se ha invertido mucho dinero, quizá sea lo que los economistas llaman *dumping*, es decir, vender algo por menos de lo que cuesta para que las empresas como Netscape, con menos base financiera, quiebren. Pero las ediciones de Windows NT son particularmente llamativas. Se ha demostrado que las dos ediciones tienen exactamente el mismo sistema operativo. Las únicas diferencias técnicas son un par de entradas en el registro de configuraciones, que afectan a algunos algoritmos, y que Windows NT Server incluye algunos programas adicionales. Microsoft había manifestado en una entrevista a Infoworld que las dos ediciones de Windows NT incluían versiones del sistema compiladas de forma distinta, mediante construcciones de C #ifdef. Peor aún, el contrato de licencia de Microsoft Internet Information Server, el servidor de HTTP de esta empresa (incluido en el "Windows NT Option Pack"), dice que no se pueden publicar pruebas sobre su velocidad sin pasar por su aprobación, lo que en mi opinión va contra el derecho fundamental de *libertad de expresión* garantizado por el artículo 20 de la Constitución Española.

Netscape es el primer navegador de buena calidad y de libre distribución para Linux, que, a su vez, es el único competidor vivo en sistemas operativos personales

Nada de esto es nuevo. Podemos recordar aquellos tiempos de Windows 3.0, en que Microsoft Excel y Microsoft Word contenían enlaces con "puertas traseras" secretas del sistema, que permitían a estos programas funcionar a pesar de un error del sistema que impedía ejecutar programas con un cierto tamaño del segmento de datos, en lugar de dar a los clientes una versión corregida de Windows, pues eso significaría reconocer públicamente un fallo y dañaría la imagen del producto. Además, tanto Windows 3.0, o Windows 3.1 como Windows 95 tienen llamadas indocumentadas. Estas llamadas pueden permitir a Microsoft escribir aplicaciones más eficientes; si bien esto siempre lo ha negado Microsoft, lo cierto es que en Windows 3.1 tanto Word como Excel usan llamadas indocumentadas, lo mismo que Internet Explorer 3.0 en Windows 95.

Por otro lado, que Netscape publique el código fuente es, en parte, el resultado de la popularidad de Linux. Éste es un ejemplo de cómo un programa desarrollado de forma cooperativa puede tener buena calidad. Además, ha propagado la idea de GNU: el código fuente público fomenta el intercambio de ideas entre programadores y, por tanto, el progreso tecnológico.

¿Cómo puede ser fiable un programa escrito por aficionados en su tiempo libre? La experiencia de muchos usuarios demuestra que Linux es un sistema bastante fiable, capaz de mantenerse encendido sin colgarse durante meses. La explicación es que cualquiera puede revisar, criticar o corregir el código. El éxito de Linux ha hecho posible que Netscape decida hacer su navegador de libre distribución. De esta forma, el programa evolucionará de forma más rápida y tendrá menos errores. Si su página de inicio pone "This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License" muchos usuarios de Windows 95 conocerán la idea del software de libre distribución. Linux ha demostrado que es una plataforma muy eficaz para la difusión de estas ideas. Pero Netscape, que es una simple aplicación dentro del escritorio de Windows 95, tendrá menos influencia en las opiniones del usuario que un sistema entero. Sin embargo, puede ayudar mucho. Netscape es el primer navegador de buena calidad de libre distribución para Linux.

Internet Explorer ya está disponible para Solaris, pero no para Linux. A pesar de que uno de los desarrolladores de la versión Unix de este programa es un defensor de Linux, parece que Microsoft ha decidido no ofrecerlo para Linux, porque este sistema es un competidor más directo que Solaris. O quizá porque un producto de Microsoft nunca tendría éxito en Linux.

Ahora los vendedores de ordenadores nuevos y los proveedores de Internet pueden dar Netscape a sus clientes. Esto frenará el avance de Internet Explorer. Pero la reacción de Netscape llega tarde, pues es muy difícil reemplazar a Internet Explorer 4.0. Además, la liberalización también tendrá efectos negativos para Netscape. Hay muchas empresas que desconfían del software desarrollado de forma cooperativa. A partir de ahora probablemente no usarán Netscape.

Pero lo que sí conseguirá Netscape es evitar que Microsoft pueda cobrar por Internet Explorer.

No creo que Netscape dure mucho tiempo. Internet Explorer es ya el navegador mayoritario, y el servidor de HTTP de Microsoft se usa mucho más que el de Netscape. Ahora bien, si Netscape quiebra, puede decidir hacer todos sus programas de libre distribución. Y eso podría hacer mucho daño a Microsoft. Entonces, cualquiera podría compilar el servidor de Netscape en un ordenador con Linux y obtener un servidor de HTTP de gran prestigio a un precio muy bajo. Podría obligar a Microsoft a bajar mucho el precio de Windows NT Server.

Muchas empresas como Netscape o Sun se quejan del doble monopolio aplicaciones/sistemas de Microsoft. Ahora bien, la industria informática se ha mostrado incapaz de ofrecer alternativas. Linux es actualmente el único competidor vivo en sistemas operativos personales; pero tiene grandes problemas de facilidad de uso y consistencia. OS/2 está prácticamente muerto: no vemos novedades sobre él. Así pues, los competidores de Microsoft tienen lo que se merecen.



JAVASCRIPT avanzado: el modelo de objetos

Hasta el momento se ha seguido una aproximación didáctica al lenguaje JavaScript basada en el ejemplo. En el presente artículo se formalizan algunos conceptos al presentar el modelo de objetos del lenguaje en sus diferentes versiones.

Cuando se comenzó a hablar de JavaScript en esta sección, se empleo el lenguaje como una extensión de HTML, añadiendo algunos manejadores de evento "aquí y allá". Ya entonces se adelantaba que una de las ventajas del lenguaje consistía en que no era necesario conocerlo para hacer uso de él y lograr interesantes efectos. A lo largo de diversos artículos, el lector ha ido profundizando cada vez más en el tema hasta llegar a manejar una serie de conceptos más avanzados, como los de jerarquías de captura de eventos y toda la terminología asociada a objetos. Ha llegado el momento ya, de formalizar el modelo de objetos de JavaScript, comentando la relación entre su generación y el código HTML y cómo el programador puede definir sus propios objetos y funciones con número indefinido de parámetros.

NUEVAMENTE LA ETIQUETA SCRIPT

La etiqueta <SCRIPT> es ya una vieja conocida de los lectores de esta sección, sin embargo, aún tiene reservados algunos secretos relacionados con las versiones del lenguaje y la existencia de scripts firmados digitalmente. Por el momento se abordará únicamente el primer tema.
Esta etiqueta admite un atributo opcional LANGUAGE que indica el lenguaje, así como su versión, en que se encuentra programado el script. Los scripts incrustados en un documento no tienen por qué estar siempre programados en JavaScript, podrían corresponder a otros lenguajes muy similares como JScript o menos similares como VBScript, ambos de Microsoft. Y aún tratando con JavaScript es importante señalar con que versión se está tratando por dos motivos:

1. Para que aquellos browsers que no entiendan la versión correspondiente a la etiqueta ignoren el script.
2. Para que aquellos browsers que sean capaces de interpretar una versión posterior a la indicada adapten su semántica a ésta. Esto es necesario ya que se da el caso en que el significado de una misma sentencia de JavaScript puede variar de una versión a otra. Sin ir más lejos, en este mismo artículo en el punto dedicado a arrays, se cita un ejemplo. Por ello, conviene especificar que versión del lenguaje se está utilizando.

Con JavaScript, las alternativas, hasta el momento, son "JavaScript", que será interpretado por Navigator 2.0 y superiores; "JavaScript1.1", que será interpretado por Navigator 3.0 y superiores y "JavaScript1.2", que será interpretado por Navigator 4.0. Por supuesto, será posible incluir scripts correspondientes a varias versiones y lenguajes en un mismo documento. Respecto a MS Internet Explorer, el autor aconsejaría no especificar versión y, simplemente, probar. Aunque la mayor parte de las extensiones de JavaScript 1.1 no se encuentran disponibles hasta la versión 4.0 de este browser.

Los scripts incrustados en un documento no tienen por qué ser necesariamente JavaScript

Otra nota a destacar es que, a partir de la versión 1.1 del lenguaje, la etiqueta <SCRIPT> admite la existencia del script en un fichero aparte. Para indicar el URL de este fichero, cuyo nombre debe terminar obligatoriamente con la extensión .js, se emplea el atributo SRC. Esto es especialmente interesante para crear librerías de funciones a compartir entre distintas páginas. Cualquier contenido que quede rodeado por la etiqueta será ignorado a menos que se produzca un error al incluir el fichero externo, por lo que suele emplearse este espacio para colocar un aviso. Por ejemplo:

```
<SCRIPT SRC="mi_script.js>
alert("No puedo cargar mi_script.js")
</SCRIPT>
```

Si se desea colocar los ficheros .js en un servidor, será necesario configurar éste asociando a esta extensión el tipo MIME application/x-javascript.

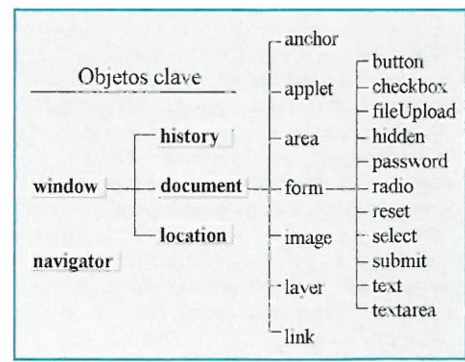
OBJETOS EN JAVASCRIPT

Cuando el cliente carga una página, genera una jerarquía de objetos con toda la información concerniente a la misma y que refleja la estructura del documento HTML. Aunque no existen clases como en Java, no está de más conocer esta jerarquía para poder utilizarla (figura 1).
Este tipo de jerarquías en terminología de programación orientada a objetos recibe el nombre de jerarquía de instancias, ya que el lugar de estar constituída por clases de objetos, lo está por instancias de objetos. En ella, cada "hijo" de un objeto es una propiedad del mismo y, como se puede apreciar, una propiedad puede ser un objeto en sí.
Para referirse a una propiedad de uno de estos objetos, será necesario cualificarla especificando el nombre de todos sus antecesores.
Por ejemplo, supóngase un documento como:

```
<TITLE>Prueba</PRUEBA>
<BODY>

<FORM NAME="form1">
<INPUT TYPE="checkbox" NAME="Entrada1"
CHECKED>
<INPUT TYPE="button"  NAME="Boton1"
VALUE="Autodestrucción">
```

FIGURA 1. JERARQUIA DE OBJETOS JAVASCRIPT.





</FORM>

</BODY>

Se definirán una serie de objetos y propiedades comunes a todo documento, como:

location.href = "http://highland.dit.upm.es:8000/prueba.html"
document.title = "Prueba"
document.fgColor = #000000

etc, y otros objetos relativos a este caso particular, como

document.form1
document.form1.Entrada1
document.form1.Boton1

con sus propias propiedades, como

document.form1.Entrada1.defaultChecked = true
document.form1.Boton1.value = Autodestrucción

El atributo LANGUAGE indica el lenguaje y la vesión de un script

Es importante conocer como se genera un documento en la pantalla del navegador a la hora de evitar errores programando JavaScript. Por regla general, los elementos de un documento se colocan en pantalla de forma secuencial según el navegador los va leyendo de la fuente HTML. Los objetos se van creando según el navegador los lee y los muestra y no antes. Por ello, sólo se podrá acceder a ellos desde JavaScript, una vez hayan aparecido. Así, si en determinado momento un script hace referencia a un objeto que aún no se ha creado, porque el documento no ha terminado de llegar, se producirá un error. Lo mismo ocurrirá si un script trata de acceder a propiedades de un objeto que aparece más tarde en el código HTML. Por ejemplo, el siguiente código produciría un error:

```
<SCRIPT>

document.write(document.mi_form.nombre.value)
</SCRIPT>
<FORM NAME="mi_form">
<INPUT TYPE="text" name="nombre"
value="echeva">
</FORM>
```

ya que cuando se llama a *document.write*, el objeto *form* aún no existe.
Es por esta razón por lo que se aconseja definir siempre las funciones en la cabecera del documento e invocarlas en el cuerpo y siempre que se haya terminado de cargar el documento. En este sentido pueden darse problemas cuando se invoca una función que se encuentra en otro frame, ya que ésta podría no haberse cargado aún.

OBJETOS CLAVE

Existe un conjunto de 5 objetos que siempre se define para todo documento, por lo que se les denomina *objetos clave* y son los siguientes:

1. navigator

Contiene propiedades para el nombre y la versión del browser WWW así como los plug-ins instalados y los tipos MIME soportados:

- *appCodeName*: Almacena el nombre en clave del browser de WWW. Por ejemplo, el de Netscape Navigator es "Mozilla"
- *appName*: Almacena el nombre del browser de WWW. Por ejemplo, el de Netscape Navigator es "Netscape"
- *appVersion*: Indica la versión del browser.
- *userAgent*: Almacena el valor de la cabecera user-agent del protocolo HTTP, que indica tabién el browser empleado. Esta propiedad es más completa ya que indica además el sistema operativo, pero es menos cómoda ya que habrá que analizarla para extraer el navegador y la versión. Por ejemplo, una conexión con Navigator 2.0 desde Windows 3.x devolvería una propiedad userAgent "Mozilla/2.0 (Win16; I)".

A partir de JavaScript 1.1, se incoporan también las propiedades:

- *mimeTypes*: Array de los tipos MIME aceptados por el browser.
- *plugins*: Array de los plugins instalados.

y el método *javaEnabled*, que devuelve *true* si el usuario tiene activado Java y *false* en caso contrario.

NOTA: existe otro método que se va a ignorar a propósito, *taintEnabled*. Tiene relación con un mecanismo de seguridad de Netscape 3.0 que ha sido abandonado por "inseguro". Por ello, se recomienda al lector que ignore cualquier bibliografía que pueda encontrar acerca de *tainting*.

2. window

Es el objeto principal y contiene las propiedades de toda la ventana. Si se ha cargado un documento frame, se genera también un objeto window por cada frame de que conste. En [Echeva-3] se dedicó un artículo al control de ventanas y frames en el que se profundizaba en este objeto.

Desde la versión 1.1 de JavaScript se pueden crear librerías de scripts

3. location

El objeto *location* representa la URL completa del documento.
Este objeto es una propiedad del objeto *window* y, aunque existe otro del mismo nombre para el objeto *document* que aparentemente se comporta igual, se recomienda encarecidamente al lector que no lo use, ya que no es exactamente lo mismo (no procede aclarar las diferencias ya que es una propiedad cuya desaparición se ha anunciado). Asimismo, aunque, en ocasiones se puede invocar directamente, conviene cualificarla siempre con la referencia a la ventana a la que pertenece lo que, en ocasiones, es obligatorio. Por ejemplo, en manejadores de evento es obligatorio indicar *window.location* en lugar de sólo *location*.
Cada propiedad del objeto location representa una porción diferente de la URL del documento. Si se recompusiera una URL a partir de estas propiedades se haría de la forma:
protocol//host:port/pathname#hash?search
Así, a continuación se explica cada propiedad y se indica el valor que tomaría para la URL *http://highland.dit.upm.es:8000/cgi-bin/prueba?a=1&b=2*

- *protocol*: Indica el comienzo de la URL, hasta los primeros dos puntos (:) incluidos. En el ejemplo: "http:"
- *host*: Almacena el nombre del host incluido su dominio, o la dirección IP del mismo. En el ejemplo: "highland.dit.upm.es"
- *port*: Representa el puerto de comunicaciones. En el ejemplo: "8000"
- *pathname*: Representa la porción de path de la URL. En el ejemplo: "/cgi-bin/prueba"
- *hash*: Almacena, si existe, el nombre del fragmento correspondiente al ancla del URL, incluyendo el carácter hash (#). Sólo tiene sentido en URLs http. En el ejemplo, estaría vacía.

Arrays de objetos

Es posible acceder a gran parte de los objetos de un documento y a través de ellos, a sus propiedades, empleando diversos arrays, algunos de los cuales ya se han estudiado en artículos anteriores: anchors, applets, arguments (explicado más adelante), elements, embeds, forms [Echeva-1], frames [Echeva-ventanas], history, images, layers [Echeva-2], links, mimeTypes, options, plugins. Con JavaScript 1.0 sólo se puede indexar los arrays con un ordinal mientras que en versiones posteriores se puede utilizar este ordinal o el nombre del objeto (si está definido). Por ejemplo, se podría aludir al primer form de un documento como document.forms[0], document.forms["mi_form"] o document.mi_form.



- *search*: Representa la query string [Echeva-3] correspondiente a la URL si existe, incluyendo el signo de interrogación. Sólo tiene sentido en URLs http. En el ejemplo: ?a=1&b=2
- *href*: Representa la URL completa, por ello, para cargar un documento es indiferente indicar location=<nueva_URL> o location.href=<nueva_URL>
- *hostname*: Equivale a la concatenación de host y port.

A veces una misma sentencia es interpretada de forma diferente según la versión de JavaScript

4. history

Se trata de un array que contiene los URLs de los documentos que se han visitado anteriormente. Como en todos los arrays, su longitud se puede consultar mediante la propiedad length aunque lo más interesante de este objeto son sus métodos:

- back: Permite retroceder al documento anterior
- forward: Permite avanzar al siguiente
- go(paso): Permite avanzar o retrocer (paso negativo) a un documento del histórico. Por ejemplo history.go(-3) volvería al antepenúltimo documento consultado.

5. document

Contiene las propiedades sobre el contenido, como colores, titulo, forms, etc. En el listado 1 se puede ver la sintaxis completa de la etiqueta HTML <BODY> que refleja gran parte de las propiedades del objeto document.

OBJETO DATE

El manejo de fechas será especialmente útil a la hora de manejar fechas de caducidad, como las que aparecen en las cookies. El tatamiento de fechas de JavaScript es prácticamente idéntico al de Java, llegando incluso a disponerse de los mísmos constructores y métodos. Como muchos otros lenguajes y sistemas contabiliza las fechas en milisegundos desde la fecha de referencia (*epoch*): 1 de Enero de 1970 a las 00:00:00. Cualquier fecha deberá ser accedida a través de un objeto *date*. Para crear uno de estos objetos se hace a través del constructor *Date* de la forma:

fecha = new Date(argumentos)

Donde los argumentos pueden ser:

- Ninguno: Toma la fecha y hora actual
- Una cadena que representa la decha según el formato: "mes dia, año horas: minutos: segundos". Por *ejemplo my_birth = new Date("August 25, 1970 15:00:00")*
- Un conjunto de enteros para el año, mes y día y, opcionalmente, horas, minutos y segundos, por ejemplo: *my_birth = new Date(70, 8, 25)* o *my_birth = new Date (70, 8, 25, 15, 0, 0)*

El objeto date cuenta con multitud de métodos, tanto para la consulta de fechas como para su ajuste. Así, para la consulta se dispone de *getMonth, getYear, getDate, getDay, getHours, getMinutes* y *getSeconds* que devuelven, respectivamente, el ordinal del mes, el año, el día del mes, el día de la semana, la hora, los minutos y los segundos. Para el ajuste, estos métodos encuentran su análogo en *setMonth, setYear, setDate, setHours, setMinutes, setSeconds*. Como se puede observar, no hay *setDay*, ya que el día de la semana se ajusta automáticamente en función de la fecha. Además de estos métodos, se cuenta con otros dos muy interesantes a la hora de obtener fechas en formato de cadaena de texto, lo que será especialmente útil para manejar cookies.:

- *toGMTString*: convierte un objeto fecha en una cadena de texto siguiendo el convenio GMT de Internet. Por ejemplo: *today.toGMTString()* devolvería algo como "Fri, 20 Feb 1998 20:00:34 GMT"
- *toLocaleString*: convierte un objeto fecha en una cadena de texto siguiendo usando el convenio local. Por ejemplo: *today.toLocaleString()* devolvería algo como "02/20/98 22:00:34". En este caso, la representación exacta dependerá del sistema local.

Otro método muy interesante, que realiza la función complementaria es:

- *parse*: Se trata un método estático de Date, por lo que siempre se invoca como *Date.parse()* aunque se disponga de instancias de objetos Date. Toma una cadena de texto con formato de fecha, como "Aug 25, 1970" y devuelve el número de milisegundos desde la fecha de referencia. Esto es muy útil para combinarlo con el método *setTime* (explicado a continuación). Lo interesante de este método es que acepta el formato de fecha estándar del IETF (Internet Engineering Task Force), "Wed, 25 Aug 1996 15:00:00 GMT +0430".

Por último, para la lectura y ajuste de fechas en el formato insterno se dispone de los métodos:

- *getTime*: Devuelve la fecha en milisegundos respecto al epoch
- *getTimezoneOffset*: diferencia en minutos ente el tiempo local y la hora GMT
- *setTime*: Ajusta un objeto data a partir de la fecha en milisegundos respecto al epoch

OBJETO MATH

El objeto Math contiene propiedades y métodos para constantes y funciones mateméticas, respectivamente. Las constates que presenta, cuyo nombre es autodescriptivo, son: E, LN2, LN10, LOG2E, LOG10E, PI, SQRT1_2 (raíz de 1/2, o 1/raiz de 2) y SQRT2. Los métodos pueden observarse en la tabla adjunta.

LISTADO 1. DEFINICION HTML DEL OBJETO DOCUMENTO

```
<BODY
  BACKGROUND="imagen_de_fondo"
  BGCOLOR="color_de_fondo"
  TEXT="color_de_texto"
  LINK="color_de_enlaces"
  ALINK="color_enlace_activo"
  VLINK="color_enlace_visitado"
  [onLoad="manejador"]
  [onUnload="manejador"]>
</BODY>
```

Al cargarse un documento se genera una jerarquía de objetos que reflejan su estructura y propiedades

Con esta función es con la que más se suele usar la sentencia with (ver cuadro). En [Echeva-2], por ejemplo, se hizo uso de este objeto en un ejemplo para programar las ecuaciones en paramétricas del movimiento de un planeta y su luna.
Todas las funciones trigonométricas toman el argumento en radianes.

OBJETO STRING

Todas las cadenas de texto son consideradas objetos *string* por JavaScript. Por tanto, sobre cualquier de ellas será posible invocar cualquier método del objeto string. Por ejemplo:

nombre = "F. J. Echevarrieta"

genera un objeto string llamado nombre sobre el que se podría invocar *nombre.toUpperCase()* para obtenerlo en mayúsculas. Paro incluso los literales son objetos, por lo que también sería válido indicar *"F. J. Echevarrieta".toUpperCase()*. La única propiedad de este objeto es *length*, que proporciona la longitud de la cadena. En la tabla adjunta se pueden ver los métodos que forman parte del objeto string. Algunos de ellos se encuentran destinados al formato de literales en HTML para su presentación con *document.write*, como *anchor*, *big*, etc. Para entender mejor el caso de *anchor*, se pondrá un ejemplo:

```
var titulo="Objetos JavaScript"
ventana2.document.writeln(titulo.anchor("titulillo"))
```

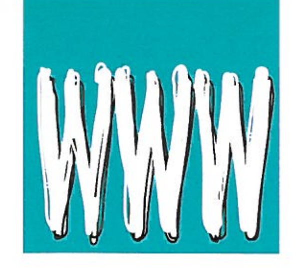


TABLA 1. METODOS DEL OBJETO MATH

METODO	DESCRIPCION
abs	Valor absoluto
sin, cos, tan	Funciones trigonométricas estándar
acos, asin, atan	Funciones trigonométricas inversas
exp, log	Exponencial y logaritmo neperiano (base e)
ceil	Redondeo a entero por defecto
floor	Redondeo a entero por defecto
min, max	Menor, mayor de dos argumentos
pow	Exponencial: Toma dos argumentos, el primero es la base, el segundo el exponente
random	Devuelve un número pseudoaleatorio entre 0 y 1. La semilla se toma de la hora actual
round	Redondeo al entero más cercano
sqrt	Raíz cuadrada

generaría:

<A NAME="titulillo">Objetos JavaScript</A>

Otros métodos, en cambio, se encargan de realizar búsquedas en cadenas y devolver subcadenas. Para entender mejor el caso de *split* se pondrá otro ejemplo:

```
array="separa esto".split(" ");
```

devolvería "separa" en *array[0]* y "esto" en *array[1]*.

FUNCIONES EMPOTRADAS

JavaScript contine algunas funciones empotradas en el lenguaje que no aparecen en forma de métodos:

- eval: Toma una cadena como argumento. Si la cadena es una expresión, devuelve el resultado de evaluarla. Si se trata de una sentencia o secuencia de sentencias, las ejecuta. El argumento de esta función puede incluir variables y propiedades de objetos. Por ejemplo, se podría realizar una función que tomara como argumentos el nombre de un elemento del primer form de un documento y un valor y asignara este último al primero de la forma :

```
function setValue(myObj, myValue) {
  eval("document.forms[0]."+myObj+".value")=myValue;
}
```

para emplearla, bastaría indicar :

```
setValue(text1,42)
```

- parseFloat y parseInt: Ambas funciones tratan de convertir una cadena en un número entero y real respectivamente. La primera, *parseFloat*, devuelve un número en punto flotante resultante de convertir la cadena de texto hasta que encuentra un carácter que no es un número, un signo, un punto o un exponente. La segunda, *parseInt*, trata de devolver un entero en la base que se le indica como argumento. Ambas funciones retornan NaN (not a number) en caso de que el primer carácter no pueda ser convertido a número.

GENERACION Y DESTRUCCION DE OBJETOS

Como se ha visto en éste y anteriores artículos, JavaScript proporciona un extenso conjunto de objetos predefinidos. Sin embargo, será también posible para el programador definir los suyos propios. Para ello, habrá que seguir los siguientes pasos:

1. Definir el constructor del objeto. Se define como si fuera una función. Por ejemplo:

```
function libro(autor, titulo, numpaginas)
{
        this.autor    = autor;
        this.titulo   = titulo;
        this.numpaginas = numpaginas;
}
```

La referencia *this* se emplea para designar al objeto que se va a crear y se le añadirá una nueva propiedad en cada asignación. En el lado izquierdo de la misma figurará el nombre de la propiedad y en el derecho, su valor. En el caso del ejemplo se ha creado una propiedad para cada parámetro de la función.

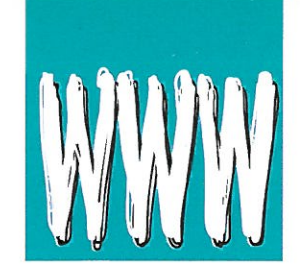
2. Se instancia el objeto. Para ello, se emplea la palabra reservada *new* acompañada del *constructor*, al que se le pueden pasar como parámetros, valores de inicio de los atributos del objeto. Por ejemplo:

libro1 = new libro("Echeva","JavaScript",193);
libro2 = new libro("Ecastro","TCL",194);

En el ejemplo se han definido dos nuevos objetos, *libro1* y *libro2*, del tipo *libro*. Estos objetos se tratarán como cualquier otro, por ejemplo *libro1.autor* devolvería el valor "Echeva". Esto ya se había hecho en anteriores ocasiones en esta sección, con la salvedad de que se instanciaban objetos predefinidos, por ejemplo, para generar nuevos objetos layer.

TABLA 2. METODOS DEL OBJETO STRING

METODO	DESCRIPCION
anchor	Crea un ancla HTML con nombre
big, blink, bold, fixed, italics, small, strike, sub, sup	Crean texto HTML formateado
charAt (posición) indicada	Devuelve el carácter de la posición
indexOf (cadena [, posición]) lastIndexOf (cadena)	Devuelven la posición de la subcadena especificada, a partir de la posición indicada o la última posición de la misma, respectivamente
link	Crea un hiperenlace HTML
split (separador)	Divide un objeto string en un array de subcadenas tomando como separador el argumento.
substring (inicial, final)	Devuelve el subconjunto de caracteres comprendido entre el inicial y el final de la cadena (se indica su ordinal)
toLowerCase, toUpperCase	Pasa la cadena a minúsculas o mayúsculas, respectivamente



DEFINICION DE METODOS

Un objeto no sólo consta de propiedades o atributos, sino también de métodos. Para que un objeto incorpore métodos será necesario definirlos en el tipo de objeto. Por ejemplo, si se quiere crear el método *muestraLibro*, que muestre las propiedades de un objeto *libro*, habría que:
1. Definir la función que actuará como método del objeto. Por ejemplo:

```
function muestraLibro()
{
        document.write("Autor : " + this.autor  + "\n");
        document.write("Título : " + this.titulo  + "\n");
        document.write("Páginas: " + this.paginas + "\n");
}
```

En el caso de métodos será necesario emplear siempre la referencia *this* para aludir a las propiedades del objeto.
2. Se incluye la función en la definición del tipo de objeto, como si se tratara de una propiedad más:

```
function libro(autor, titulo, numpaginas)
{
        this.autor    = autor;
        this.titulo   = titulo;
        this.numpaginas = numpaginas;
        this.muestraLibro = muestraLibro;
}
```

Con esta definición del tipo *libro*, una llamada a *libro1.muestraLibro()* mostraría el valor de todos los atributos de *libro1*.
Puede suceder que en determinado momento se desee añadir una propiedad a un objeto. Para ello, bastará realizar una asignación como si la propiedad ya existiera. Por ejemplo:

```
libro1.ISBN = "84..."
```

PROPIEDADES OBJETO

En numerosas ocasiones se ha visto como las propiedades de un objeto podían ser, a su vez, objetos, siendo uno de los casos más curiosos el que aparecía con los objetos *layer* y *document* ya que cada uno de ellos podía ser propiedad del otro. Lo mismo ocurrirá con los objetos que defina el programador. La forma de definir al constructor del objeto principal será la misma, pero, a la hora de instanciarlo, uno o varios de sus parámetros, en lugar de ser cadenas de texto o números, será una referencia a objeto. Por ejemplo, si se crea un objeto editorial:

```
function editorial(nombre, ciudad)
{
        this.nombre = nombre;
        this.ciudad = ciudad;
}
```

y se emplea esta nueva definición de libro:

```
function libro(autor, titulo, numpaginas, editorial)
        this.autor    = autor;
        this.titulo   = titulo;
        this.numpaginas = numpaginas;
        this.editorial  = editorial;
}
```

se podría hacer:

```
mi_editorial = new editorial("Prensa Técnica", "Madrid");
mi_libro = new libro("Echeva", "JavaScript", 193, mi_editorial);
```

Ahora, si se desea saber el nombre de la editorial de *mi_libro*, se podría indicar como *mi_libro.editorial.nombre*.

La sentencia for ... in

Esta sentencia produce un bucle for en el que se realiza una iteración por cada propiedad de que disponga un objeto. Su sintaxis es:

```
for (variable in objeto)
{
  sentencias
}
```

En cada iteración, variable toma el nombre de una propiedad del objeto y se ejecutan las sentencias. Así, es muy simple definir una función que muestre todas las propiedades con las que cuenta un objeto, así como sus valores:

```
function dump_props(obj, obj_name) {
   var result = ""
   for (var i in obj) {
      result += obj_name + "." + i + " = " + obj[i] + "<BR>"
   }
   result += "<HR>"
   return result
}
```

AMPLIACIONES A LA VERSION 1.0

Hasta ahora se ha visto cómo es posible añadir nuevas propiedades a objetos ya creados, simplemente, asignando un valor a una propiedadad que aún no tienen. Sin embargo, esto sólo afecta a aquel objeto concreto sobre el que se ha realizado la asignación. Así, en el ejemplo propuesto, *libro1.ISBN = "84..."*, se añadiría una propiedad *ISBN* a *libro1*, pero no a *libro2* ni a los objetos *libro* que se crearan a partir de ese momento. Con JavaScript 1.1, a partir de Navigator 3.0, es también posible crear la propiedad para todos los objetos de un mismo tipo que se creen a partir de ese momento. Para ello, se emplea la nueva propiedad prototype del tipo. Por ejemplo:

```
libro.prototype.ISBN = null
```

La propiedad *prototype* se encuentra disponible en todos aquellos objetos JavaScript susceptibles de ser instanciados mediante *new*, incluyendo los predefinidos, como *layer*.
Asimismo, con la versión 1.1 del lenguaje se puede acceder al constructor de un objeto y su implementación, a través de la propiedad constructor. Por ejemplo, *libro1.constructor*.
Una última incorporación de la versión 1.1 de JavaScript, es la posibilidad de destruir objetos. Para ello, bastará con igualar su referencia a *null*.
Con Navigator 4.0, se inluye la versión 1.2 del lenguaje, que introduce una nueva forma de crear objetos empleando notación literal de la forma

```
nombre_obj = {propiedad1:valor1, propiedad2:valor2 ...}
```

por ejemplo:

```
libro = { autor: "Echeva", titulo: "JavaScript", numpaginas: 193,
mi_editorial: { nombre: "Prensa Técnica", ciudad: "Madrid" }}
```

Sentencia with

Gracias a la sentencia with, se puede evitar tener que cualificar siempre las propiedades de un objeto con el nombre del mismo en un conjunto de sentencias. Su sintaxis es :

```
with (objeto)
{
  sentencias
}
```

e indica que todas las variables y funciones que aparezcan en el conjunto de sentencias se buscar·n primeramente entre las propiedades y mÈtodos del objeto. Esto se suele usar mucho para operaciones matem·ticas. Por ejemplo, en

```
var a, x, y
var r=10
with (Math) {
  a = PI * r * r
  x = r * cos(PI)
  y = r * sin(PI/2)
}
```

se asume Math.PI, Math.cos y Math.sin

Obsérvese que, en este ejemplo, se ha definido un objeto propiedad de otro, en la misma línea. Este tipo de sintaxis se pude utilizar sólo una vez, cuando la página HTML se carga.

ARRAYS Y ARRAYS ASOCIATIVOS

Existe una forma alternativa de acceder a las propiedades de los objetos, empleando una notación de array como *libro["autor"]*, en lugar de *libro.autor*. Este tipo de arrays recibe el nombre de arrays asociativos ya que cada elemento índice se asocia con una cadena literal. En los artículos dedicados a layers se vio lo importante que podía llegar a ser, cuando la propiedad en sí era un objeto layer variable que se pasaba como parámetro a una función.
La primera versión de JavaScript no incorpora arrays en el lenguaje, por lo que se empleaba este mecanismo para simularlos. Así, era común simular arrays definiendo propiedades de un objeto empleando números para nombrarlas. Por ejemplo: *casilla[1] = 0*. Esto es lo que ocurre con la versión 3.0 de MS Internet Explorer, ya que este browser no ha incorporado los arrays hasta la versión 4.0.
A partir de JavaScript 1.1, se incorpora el objeto *Array*, que puede ser instanciado mediante una de las siguientes formas:

1. *Nombre_del_array = new Array()*
2. *Nombre_del_array = new Array(longitud)*

donde longitud es la longitud inicial del array. Es importante resaltar lo de "inicial" ya que si se utiliza un elemento fuera de rango, el array crece dinámica y automáticamente hasta alcanzar el tamaño necesario. Así:

```
demo=new Array()
demo[99]="hola"
```

genera un array de 100 elementos, numerados del 0 al 99 y todos iguales a *null* (valor cuando no se inicializan) salvo el 99. La longitud del array se encuentra reflejada por la propiedad length del mismo.
También será posible crear arrays multidimensionales, para lo que habra que crear arrays cuyos elementos sean arrays. Por ejemplo, el siguiente código genera una matriz de 5x5 y la inicializa a unos con objeto de que el lector vea la notación empleada:

```
matriz = new Array(5)
for (i=0; i<5; i++) {
  a[i] = new Array(5)
  for (j=0; j<5; j++) {
    a[i][j]=1
  }
}
```

Es posible asignar valores iniciales a todos los elementos de un array a la vez. Por ejemplo:

iniciado = new Array("Hola", variable, 35)
Aquí se produce una ambig¸edad de notación: øqué se debería interpretar entonces por *nuevo = new Array(1)*? Según lo expuesto, podría estar definiéndose un array de un elemento o un array de un elemento inicializado a 1. Pues bien, las versiones 1.0 y 1.1 de JavaScript adoptarán la primera interpretación, mientras que la versión 1.2 adoptará la segunda.
En definitiva, si se desea emplear arrays en versiones anteriores a la 3.0 de Navigator y a la 4.0 de Explorer, será necesario simularlos mediante objetos. En otro caso, se podrá utilizar directamente el objeto Array.

EL OBJETO FUNCION Y EL ARRAY ARGUMENTS

JavaScript define un objeto para cada función que crea el usuario y representa sus argumentos en un array al que llama arguments. Gracias a este array, se puede pasar a una función un número de parámetros mayor de aquel con fue declarada.
Para ello, es posible consultar los argumentos que se le pasan en cada invocación empleando notación de arrays. Así, cada parámetro puede ser accedido dentro de la función mediante el array arguments de la forma:

nombre_de_la_función.arguments[i]

donde, como en todo array, i oscila entre 0 para el primer parámetro y el valor de la variable *arguments.length* para el último

Por ejemplo, la función crea_lista del **listado 2** genera listas HTML con un número indefinido de argumentos, aunque únicamente exige un argumento que especifica el tipo de lista: *u*, para no numerada y *o* para numerada. Así, un ejemplo de llamada sería:

crea_lista("u","patatas","melones","uvas","solomillo")

LISTADO 2

```
function crea_lista(type) {
  // Etiquetas de comienzo de lista
  document.write("<" + type + "L>")

  // Bucle de recorrido de los argumentos
  for (var i = 1; i < list.arguments.length; i++)
    document.write("<LI>" + list.arguments[i])

  // Etiqueta de fin de lista
  document.write("</" + type + "L>")
}
```



CURSO DE ENSAMBLADOR

Nivel: Iniciación

Capítulo 13º

Autor: Enrique de Alarcón

Ensamblador en los micros 80386

En el presente capítulo se da por concluido todo lo relativo al ensamblador del microprocesador 8086. A partir de este número se continuará el curso con la explicación de todo lo relacionado con el ensamblador de las generación de chips 386 y superiores, lo que incluirá muchos conceptos nuevos que ayudarán al lector a entender, mejor incluso, el funcionamiento interno de los actuales sistemas operativos.

El mes pasado, se acabó de detallar todo lo que quedaba por ver del ensamblador del procesador 8086. Pero la realidad es que este microprocesador tiene muchos años y, actualmente, todo el mundo programa usando el ensamblador de los modelos 80386 de la familia x86 y superiores, que destacan porque incorporan una gran cantidad de innovaciones que convierten a nuestra máquina en un sistema mucho más potente, como es un mayor rango de memoria física direccionable, incorporación de más registros, aumento de tamaño de los ya existentes, sistemas hardware avanzados de gestión de memoria, multitarea y la incorporación, entre otras cosas, del coprocesador matemático en el propio chip (a partir del 486DX) y que nos da todo un set de instrucciones destinadas a cálculos con números reales y enteros, lo que nos permite dar la potencia de una calculadora científica a nuestros programas ensamblador.

Seguramente, a partir de ahora, el curso se hará mucho más interesante para aquellos que usan o están interesados en aplicar el lenguaje ensamblador como complemento a sus propios programas (por ejemplo) de C o C++, lo cual es una práctica muy habitual, sobre todo entre los programadores de videojuegos y aplicaciones de altos requerimientos de procesado.

El ensamblador del 80386 y superiores es de 32 bits

Para empezar esta nueva etapa del curso, en el presente capítulo se van a detallar todos los nuevos conceptos asociados a la arquitectura interna del 386.

Antes de empezar, sólo recordar que para preguntas al autor, el lector puede mandar un e-mail a la dirección *erde@arrakis.es* o hacerlo desde la web: *www.arrakis.es/~erde.*

FIGURA 1. MAPA DE LOS REGISTROS INTERNOS DISPONIBLES EN UN 80386.

ESTRUCTURA INTERNA DE LOS REGISTROS EN EL 386

ESQUEMA DE LA ESTRUCTURA DE UN REGISTRO CONVENCIONAL

Registro 32 bits EAX

Palabra AX

byle D								byle C								Byte AH								Byte AL							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

bit

REGISTROS GENERALES

	31- -15-	-0
EAX		AX
EBX		BX
ECX		CX
EDX		DX

REGISTROS GENERALES

CR0, CR1, CR2, CR3 (31- -0)

REGISTROS DE SEGMENTO

SS, CS, DS, ES, FS, GS (15- -0)

REGISTROS PUNTERO

	31- -15-	-0
EIP		IP
ESI		SI
EDI		DI
ESP		SP
EBP		BP

REGISTRO DE BANDERAS

--	--	--	OF	DF	IF	TF	SF	ZF	--	AF	--	PF	--	CF
-	-	-	1	2	3	4	5	6	-	7	-	8	-	9

PROCESADOR DE 32 BITS

Para poder direccionar más memoria y operar con valores mayores, estos procesadores están fabricados con tecnología interna de 32 bits, lo que conlleva varias consecuencias.

Por un lado, significa que todos los registros internos de la CPU, tanto los generales, los de puntero y demás, ahora poseen un tamaño de 32 bits, por lo que pueden contener valores entre 0 y 4294967296 (2^32). Por otro, al ser registros de memoria de 32 bits se ha podido dotar al sistema de un bus de datos de 32 bits, con lo que ahora se pueden direccionar 4 Gbytes de memoria (2^32).

Ahora los registros pasan a tener de 16 a 32 bits

Como es lógico, para poder aprovechar los registros de 32 bits, todo el set de instrucciones del ensamblador se ha ampliado para poder operar con ellos, teniendo así que podemos sumar números de 32 bits, multiplicarlos, desplazarlos, mover cadenas en bloques de 4 bytes simultáneos, etc...

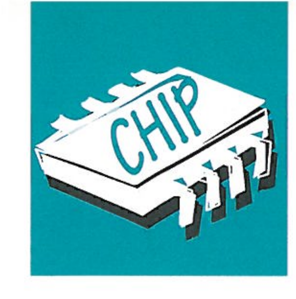
LOS REGISTROS EN EL 386

Como hemos señalado, todos los registros se han ampliado en tamaño, si bien los registros de 16 bits del 8086 continúan existiendo como tales para conservar la compatibilidad con los programas 386. Para hacerlo posible, se optó por la solución más simple: añadir un bloque de 16 bits a cada uno de los registros de 16 bits del procesador, que se han dejado como palabra baja de cada uno de los nuevos registros.

Para diferenciar entre registros completos de 32 bits y las palabras bajas que hacen la función de registros de 16 bits, simplemente se añade una E delante del nombre de cada nombre de registro para indicar que nos referimos a todos los 32 bits completos (*E* de *E*xtended).

Dicho todo esto, tenemos pues que, ahora, disponemos de los registros llamados EAX, EBX, ECX, EDX, ESI, EDI..., cada uno de los cuales está formado por dos palabras de 16 bits, de las cuales, la inferior corresponde al registro de 16 bits que se obtiene de quitar E al nombre, por lo que, por ejemplo, EBX son dos palabras de la cual, los 16 bits inferiores corresponden al registro BX que, a su vez, se puede continuar dividiendo, accediendo como se hacía en el 8086, en dos registros tipo byte (BH y BL).

Además de haberse doblado en tamaño, los diseñadores del procesador también incorporaron algún registro nuevo no presente en el 8086 y que son los llamados FS y GS y que, como puede



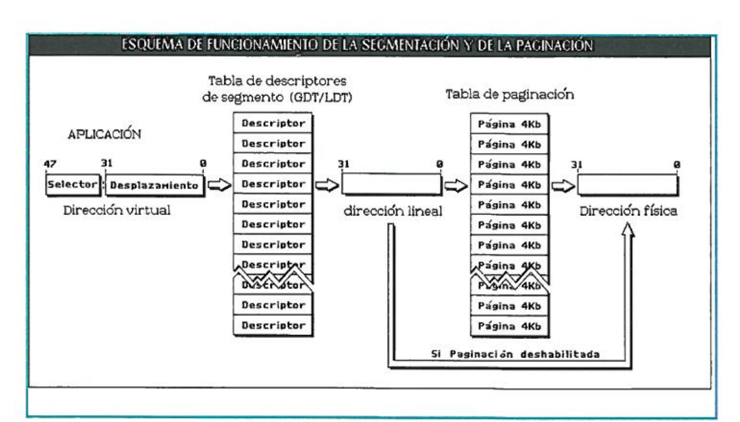


FIGURA 2. ESQUEMA DE LAS ETAPAS POR LAS QUE PASA UN PUNTERO DE MEMORIA VIRTUAL HASTA CONVERTIRSE EN UNA DIRECCION DE MEMORIA FISICA

deducirse por los nombres, son registros de segmento que resultan de gran utilidad, puesto que con los dos con los que se disponía antes para crear código de aplicaciones el programador se sentía muy limitado y obligaba a aumentar la complejidad de las rutinas del ya de por sí complejo lenguaje.

En el 386 se han incorporado dos avanzados sistemas de gestión de memoria

En la fotografía número 1 el lector puede ver un esquema de todos los registros que posee un 386.

GESTION AVANZADA DE MEMORIA

Como se acaba de decir, al microprocesador se le han incorporado dos avanzados sistemas de gestión de memoria por hardware. Uno es la <*segmentación de memoria*> y el otro es el conocido como <*paginación de memoria*>.

Estos dos modos de gestión son complementarios, es decir, se pueden usar conjuntamente, haciendo que primero se ejecute, por así decirlo, la segmentación y después, si está activada, la paginación.

Cuando ambos están activos dan la capacidad al sistema de poder usar memoria virtual y la de poder proteger la memoria de forma eficiente e inviolable, de forma que, por ejemplo, una tarea A no pueda acceder a los bloques de memoria asignados a una tarea B, hecho que es un requisito indispensable en un sistema multitarea.

DIFERENCIAS FRENTE AL MODO REAL

Aparte de que se puede direccionar más memoria en el llamado modo protegido que en un 8086, la forma de crear punteros también cambia un poco. Si el lector recuerda, hasta ahora los punteros se creaban siempre mediante la combinación de dos elementos, un segmento y un desplazamiento (SEG:OFFSET), y para realizar la conversión de puntero a dirección de memoria física, simplemente tenía que aplicarse la fórmula ((SEGMENTO*16) + DESPLAZAMIENTO).

Pero ahora con el modo protegido todo ha cambiado, y aunque los punteros se siguen realizando usando un registro de segmento con otro de desplazamiento, los segmentos ahora no se consideran como tales sino que han pasado a ser lo que se conoce como *Selector*.

Un selector es sólo un indicador de número de elemento, funciona como un puntero dentro de una tabla, y ello es así porque en el nuevo modo de gestión de *segmentación* implementado en la CPU, cada bloque de memoria del sistema está registrado en una tabla, en la tabla de segmentación, y de ahí que ahora cuando se hace un puntero dentro de un bloque de memoria, lo que realmente se indica en el puntero es el BLOQUE definido + DESPLAZAMIENTO dentro de ese bloque.

En la fotografía número 2 el lector puede ver un mapa-esquema donde observar el funcionamiento de la gestión de memoria que realiza la CPU por hardware. La explicación de lo esquematizado es muy simple.

Una aplicación cualquiera (X), ejecuta una instrucción que pretende realizar un acceso a una dirección de memoria mediante un ***Selector:Desplazamiento.*** Dada la petición por parte de la instrucción, la CPU examina si el segmento número *Selector* dentro de la tabla de Segmentación es válido (mira sus atributos de estado, permiso de escritura y lectura, etc...) y tras este proceso de filtrado se obtiene la llamada *dirección de memoria lineal.*

Una vez llegados a este punto, tenemos que si la paginación de memoria está desactivada, entonces tenemos que la dirección lineal de memoria obtenida corresponde ya con la dirección física de memoria. Por el contrario, si la paginación está activada, la CPU somete esta dirección a otro proceso llamado de *traducción de páginas* que una vez concluido nos dará la tan deseada dirección de memoria física.

SEGMENTACION DE MEMORIA

La segmentación hace que la memoria esté dividida en bloques que quedan registrados en una tabla que tiene localizada la CPU, la llamada tabla de segmentación.

Para examinar si un puntero *X:Y* es válido, la CPU solamente tiene que examinar si el segmento definido como el elemento *X* dentro de la tabla de segmentación posee los atributos correctos y si el desplazamiento *Y* no excede su tamaño.

Por cada segmento que se define en el sistema, en la tabla de segmentación solamente se necesita guardar un valor de 64 bits que en argot técnico se conoce por *Descriptor*.

Un descriptor internamente no tiene ningún misterio. Está dividido básicamente en tres subcampos de información: la dirección base (4 bytes), el tamaño del bloque definido o límite (2 bytes) y los atributos relativos de estado, como son los permisos de lectura y escritura, el nivel de privilegio que posee, etc...

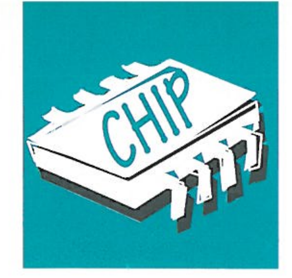
Las CPU de 32 bits pueden funcionar también como un 8086

Para que ningún lector pueda tener alguna sorpresa mirando algún libro o manual técnico, avisamos que dentro de las tablas de segmentación nos podemos encontrar tres clases diferentes de descriptor que son: <*segmentos de memoria*>, <*sistema*> y <*puertas*>.

Para simplificar su explicación, y dado que aclarar todo lo que significa cada uno de los atributos de cada tipo de descriptor ocuparía un artículo entero, se ha hecho un esquema de los tres tipos de descriptor existentes en la fotografía número 3, donde se puede observar la posición de cada componente dentro del

Nuevos modos de funcionamiento

Como ya habrá oído hablar el lector seguramente en muchas ocasiones, los procesadores 386 y superiores disponen de varios modos de funcionamiento que permiten hacer que la máquina pueda funcionar como si fuese un 8086 (el llamado modo real virtual) o hacer que funcione como un chip de 32 bits reales (el modo protegido), modo que conjuntamente con dos avanzados sistemas de gestión de memoria implementados, hace que la máquina pueda ejecutar programas en paralelo y hacer realidad la ejecución multitarea de la que tanta gala hacen los actuales sistemas operativos como Windows 95 o Windows NT.



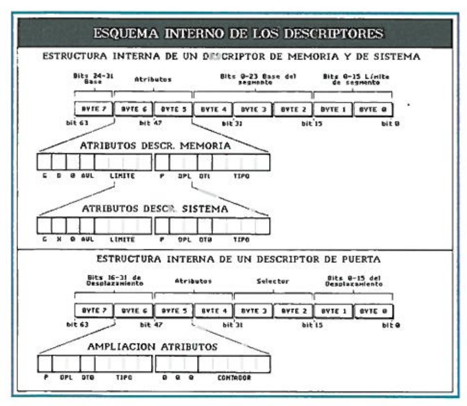


FIGURA 3. ESQUEMA CON LA ESTRUCTURA INTERNA DE LOS TRES TIPOS DE DESCRIPTOR QUE PODEMOS ENCONTRAR EN LAS TABLAS DE SEGMENTACION

mismo así como del contenido de la zona de atributos de cada tipo.

LA PAGINACION DE MEMORIA

La segunda etapa en la conversión de direcciones virtuales a físicas, la llamada paginación de memoria (traducción de páginas), simplemente se encarga de convertir las direcciones lineales en físicas basándose, también para ello, en una tabla que posee la CPU localizada en memoria.

Gracias al sistema de paginación, la CPU puede usar memoria virtual

La diferencia principal entre la paginación frente a la segmentación es que mientras en la segmentación se controlan bloques de memoria de tamaño variable, en la paginación se gestionan bloques de tamaño fijo de 4 Kbytes siempre que se conocen por el nombre de páginas, y de ahí que a este sistema se le conozca por el nombre de *paginación.*
El mecanismo de funcionamiento de la paginación es fácil de comprender. El microprocesador posee una tabla localizada donde cada página lineal de memoria posee un equivalente en las páginas físicas en las que se ha dividido toda la memoria.
Por defecto, cada página lineal posee asociada la página física que ocupa su misma dirección de inicio, que posee la misma localización física. Pero si se piensa bien, solamente cambiando las equivalencias que hay en las tablas, podemos hacer, por ejemplo, que haya páginas físicas que no sean accesibles cuando el sistema quiera, que es igual a decir que se pueden proteger páginas determinadas de ser leídas y escritas cuando se quiera, y al ser un mecanismo hardware, no del sistema operativo, es un método eficaz del todo e inviolable por ninguna aplicación que esté por debajo del sistema operativo.
Además de ello, es también un sistema perfecto para poder crear sistemas con memoria virtual, ya que podemos definir páginas que no estén realmente en la memoria y que cuando haya petición de llamada a ellas, éstas se carguen en una zona temporal de memoria (de intercambio), lo que da la ilusión al sistema de que hay más memoria instalada que la realmente existente.

REGISTROS DE CONTROL DE LA CPU

Como el 386 posee muchos parámetros de funcionamiento y estado, consecuencia de su complejidad interna, los diseñadores del chip unieron todas las banderas de control y lo colocaron todo en unos registros especiales.
En total hay cuatro registros, llamados CR0, CR1, CR2 y CR3, respectivamente, y sus letras son la abreviación del inglés <*Control Register*>.
Cada uno de estos registros tiene un tamaño de 32 bits y para leer y escribir en ellos simplemente tenemos que realizar un <MOV> que pase su contenido a un registro general o viceversa.
De los cuatro registros, CR2 y CR3 los usa el mecanismo de paginación, y de CR0 y CR1, los bits 30 a 5 del primero y los 11 a 0 del segundo no contienen nada útil, por lo que deberían estar siempre a cero.

LOS REGISTROS CR0 Y CR1

De CR0 hay cuatro bits que se usan para el control del coprocesador matemático, que son:
- **Bit ET:** Este proviene del inglés <*Extension Type*> e indica qué protocolo usará el 386 para comunicarse con el 387. ET=1 indica al sistema que hay un 387 instalado, mientras que ET=0 indica lo instalado es un 287.
- **Bit TS:** Este flag proviene del inglés <*Task Switched*> (Tarea conmutada) y lo usa la CPU para acelerar el cambio de tarea activa en algunos casos (en sistemas multitarea, por supuesto).
- **Bit MP:** Proviene de <*Matematic Present*>, controla si la instrucción *Wait* producirá un error o no al ejecutarse (una excepción).
- **Bit EM:** Que viene del inglés <*EMulate*> y lo usa la CPU para controlar si las instrucciones del 387 deberán generar una excepción (interrupción) o si deberán ejecutarse.

El mecanismo de que se puedan generar excepciones automáticamente cada vez que se intenta ejecutar una instrucción del coprocesador es fundamental para poder crear los famosos emuladores del 387 que todavía circulan, ya que estos se basan en estas excepciones para entrar en funcionamiento.
Hay también en CR0 dos bits (dos *flags*) que son los que controlan el estado de los sistemas de segmentación y de paginación, y son los siguientes:
- **Bit 0, PE:** Si este bit está a 1 significa que la segmentación está activada y, en consecuencia, que el procesador está funcionando en modo protegido (ya que es obligatorio de este modo). Por otro lado, si este bit está a cero significa que no hay mecanismos avanzados de gestión de memoria activos y que, por lo tanto, el procesador está funcionando en modo real (como si de un 8086 se tratara).
- **Bit 31, PG:** Si este *flag* vale 1 significa que la paginación está activada. En caso de valer 0 significa que está deshabilitada, aunque con ello no se puede determinar si el procesador está en modo protegido o no, dado que como he dicho ya, la paginación es sólo opcional.

En el ensamblador del 386 existen 6 registros internos añadidos

LOS REGISTROS CR2 Y CR3

Estos dos registros los usa el procesador para controlar los mecanismos de paginación y segmentación. En CR3 está almacenada siempre la dirección física de la página que contiene la tabla principal de paginación.
Por otro lado, CR2 lo usa para registrar en él el último error de excepción que se haya producido durante los procesos de paginación.

NIVELES DE PRIVILEGIO

Además de los sistemas de gestión de memoria que incorpora el 386, gracias a los cuales se hace posible la multitarea, existe también como

Para los más curiosos

Para aquellos que siempre quieren saber todo sobre lo que leen, decir que no sólo hay una tabla de segmentación, sino que existe una tabla principal, la llamada <Tabla Global de Descriptores> (en inglés GDT) y después tenemos que cada aplicación que esté funcionando en el sistema posee asociada una de propia y local con los segmentos de uso exclusivo para ella y que no son accesibles desde los otros programas. A la tabla que posee cada aplicación se la llama <Tabla de Descriptores Local> (en inglés LDT).
Asimismo, tampoco existe una sola tabla de paginación con una lista de todas las páginas disponibles, sino que hay una tabla principal de entrada que hace las veces de índice de todas las demás que haya, mecanismo que se usa para ahorrar memoria y no tener que tener una gran cantidad de memoria destinada a almacenar equivalencias de páginas.



añadido un sistema de protección a nivel interno de las aplicaciones que hace que a todos los segmentos que haya en su interior puedan poseer niveles de privilegio (o preferencia por así decirlo). En total hay cuatro niveles de privilegio posibles, que van del 0 al 3, y de los cuales, el nivel cero es el que posee un mayor nivel de privilegio, y el tres, en consecuencia, el que menor tiene.

En modo protegido no se habla de segmentos si no de selectores

Además de lo dicho, junto con este sistema de capas tenemos que el 386 posee un conjunto de instrucciones englobadas dentro de las llamadas de <*sistema operativo*> que sólo pueden ser ejecutadas desde un nivel de privilegio 0.
¿Para qué todo esto? Pues muy simple; gracias a esto, puede protegerse a los sistemas operativos de que las aplicaciones puedan alterar o modificar el funcionamiento de la CPU y los sistemas de protección de memoria, con lo que el Sistema siempre está protegido ante las aplicaciones y, de esta forma, nunca puede perder la seguridad del sistema y, por tanto, su estabilidad.
Pongamos un ejemplo, el sistema Windows 95. Cuando arranca Windows 95, todo el kernel del sistema operativo se almacena en bloques de memoria que poseen un nivel de protección cero. Por lo que podrá ejecutar las instrucciones de S.O. sin producir error alguno. Después, a todas las aplicaciones que carguemos, gracias a que existen 4 niveles de protección, se les da niveles inferiores de privilegio, con lo que tenemos, de esta forma, que las aplicaciones Windows 95 no pueden realizar operaciones comprometidas para el sistema y que, por lo tanto, Windows siempre puede mantener el control del sistema frente a los programas.
De no ser por esta capacidad, pese a la existencia de los mecanismos de protección de memoria que posee el 386, estos no servirían de nada dado que cualquier aplicación podría acceder a las tablas de segmentación y paginación, con lo que no habría ninguna seguridad en el sistema.
Explicado esto, se puede deducir el porqué es tan difícil que una aplicación bajo Windows 95 o NT pueda bloquear el sistema y dejarlo inutilizado.

INSTRUCCIONES DE CONTROL DE LA CPU

Para poder gestionar las tablas de segmentación, las de paginación, para poder cambiar modos de funcionamiento y demás operaciones similares, se han incorporado en el ensamblador del 386 toda una serie de instrucciones destinadas a tal fin. Son las comúnmente clasificadas como instrucciones de Sistema Operativo, ya que sólo se pueden ejecutar desde un nivel de privilegio 0 (el que posee normalmente el S.O.), y en total son 18. A continuación, se explican un poco por encima algunas de ellas para que el lector vea cómo funciona internamente la CPU, aunque el propósito no es que el lector llegue a aprender a utilizarlas, ya que sólo las puede usar el sistema operativo.

• **ARPL** ***op1,op2***: Comprobar nivel de privilegio (RPL).
• **LAR** ***op1,op2***: Cargar en el primer operando los atributos del descriptor de segmento indicado en el operador 2.
• **LGDT** ***op1***: Carga la tabla de descriptores indicada en Op1, que es un puntero completo de 48 bits (base:límite).
• **LIDT** ***op1***: Es igual a la anterior pero sirve para cargar la tabla de descriptores de interrupción.
• **LLDT** ***op1***: Esta instrucción es igual a LGDT pero con la tabla local de segmentación
• **LSL** ***op1,op2***: Carga en el primer operando el límite del descriptor de segmento apuntado por el selector indicado en el operador 2.
• **LTR** ***op1***: Carga en el *registro de tarea* el selector *Op1*.
• **SGDT** ***op1***: Almacena en memoria el puntero donde está localizada la tabla de descriptores global GDT.
• **SIDT** ***op1***: Almacena en memoria el puntero donde está localizada la tabla de descriptores global de interrupciones.
• **SLDT** ***op1***: Idéntica en funcionamiento a SGDT pero con la tabla local.
• **VERR** ***op1***: Detecta si el segmento correspondiente al selector contenido en el operador 1 permite la lectura.
• **VERW** ***op1***: Detecta si el segmento correspondiente al selector contenido en el operador 1 permite escritura.

EJEMPLOS DE CODIGO

Muchos se preguntarán si la programación en ensamblador para 32 bits diferirá mucho de lo que se ha visto hasta ahora pero, como ya se ha dicho, básicamente se limita a que ahora se pueden usar registros de 32 bits. A continuación se dan algunos pequeños ejemplos de código de 32 bits.

Gracias a la FPU el ensamblador adquiere una gran capacidad de cálculo

1-Ejemplo de una multiplicación con signo de 32 bits:

```
mov eax,ELEMENTO_1
mov ebx,ELEMENTO_2
imul ebx
mov RES_ALTO,edx
mov RES_BAJO,eax
```

2-Ejemplo de una división sin signo de un elemento de 64 bits entre otro de 32:

```
mov edx,ELEMENTO_1A
mov eax,ELEMENTO_1B
mov ebx,DIVISOR
div ebx
mov RES,eax
```

3-Ejemplo del limpiado de un bloque de 128 Kbytes de memoria.

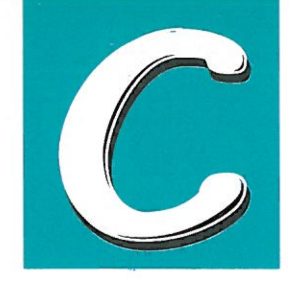
```
mov ax,SEGMENTO
mov es,ax
sub eax,eax
mov edi,eax
mov ecx,16000
rep stosd
```

Bibliografía

- **Cómo programar en ensamblador 80x86. Biblioteca técnica de Programación. Prensa Técnica.**
- **8088-8086/8087 Programación ENSAMBLADOR en entorno MS-DOS, Miguel Ángel Rodríguez Roselló. Anaya Multimedia.**
- **Programación del 80386/387. Manual de referencia y técnicas avanzadas de programación para diseñadores de sistemas, programadores y usuarios avanzados. John H. Crawford/Patrick P. Gelsinger. Anaya Multimedia.**
- **80386 Data sheet. Intel Corporation**
- **80386 Assembly Languaje Reference Manual. Intel Corporation.**
- **80387 Data Sheet. Intel Corporation.**

Próximos capítulos

Para los lectores que deseen saber qué aparecerá en los próximos capítulos, decir que se explicarán las instrucciones nuevas que aporta el 386 al set del 8086, se aprenderá a usar las instrucciones del coprocesador matemático, se mostrará como crear rutinas ASM para C (como por ejemplo el Watcom C/C++), etc...



Autor: Fernando de la Villa

POO: Las Clases [I]

Las clases constituyen el elemento fundamental de la programación orientada a objetos con C++. En este número, da comienzo una pequeña serie dedicada por entero a tratar sus características.

Las clases de C++ se crearon para poder construir tipos de datos definidos por el usuario que fueran tan fáciles y cómodos de manejar como los tipos predefinidos. Pero no sólo son tipos de datos. Además, permiten plasmar en la práctica los conceptos de la programación orientada a objetos. Es un tema amplio y, hasta cierto punto, complejo, debido al gran número de características que presentan estos elementos. Por ello, a partir de este mes comienza una pequeña serie en la que serán tratadas las clases en profundidad.
Es fundamental que al lector le quede claro todo lo relacionado con las clases para poder escribir programas en C++ con soltura. Se recomienda releer el texto tantas veces como se considere necesario y practicar, en la medida de lo posible, con un buen compilador de C++.

ESTRUCTURA BASICA DE UNA CLASE

Como ya se indicó en la primera entrega del curso, una clase está formada por unos datos y las operaciones permitidas sobre esos datos. Pero el acceso a esos miembros está limitado para poder proteger la integridad de la información almacenada. Existen tres tipos de acceso:

- Público.
- Privado.
- Protegido.

Los miembros públicos pueden ser accedidos tanto desde el interior como desde el exterior de la clase. Sin embargo, los miembros privados sólo son accesibles desde el interior de la clase. Por último, los protegidos se comportan como los privados, pero pueden ser vistos por otras clases bajo determinadas condiciones que se verán más adelante.
Además, existe un tipo especial de funciones (funciones amigas) que permiten saltarse estas restricciones.
Por ejemplo, una clase se puede ver como un edificio con tres pisos, cada uno de ellos con una puerta. La puerta del primer piso siempre está abierta y, por tanto, cualquiera puede pasar y ver lo que hay dentro (acceso público). Sin embargo, la puerta del segundo tiene una cerradura. Sólo aquellos que poseen la llave adecuada pueden pasar. Esta llave se da a los inquilinos del edificio y a determinadas personas ajenas al mismo (acceso protegido). La puerta del último piso también posee una cerradura, pero la llave sólo la pueden tener los inquilinos del edificio (acceso privado).
Según lo anterior, la estructura básica de una clase sería:

```
class <nombre_clase> {
  private:
    <lista_miembros_privados>
  protected:
    <lista_miembros_protegidos>
  public:
    <lista_miembros_públicos>
} <lista_de_objetos>;
```

Las secciones *private, protected* y *public* son opcionales. Asimismo, la lista de objetos también puede ser omitida (aunque no se debe olvidar el punto y coma final).
Si una lista de miembros no está precedida por uno de los tres especificadores de acceso, se asume *private*. Por ejemplo, estas dos definiciones de clase son equivalentes:

```
class figura {
  int x, y;      // Atributos privados
  int color (); // Método privado
  public:
    void dibujar (); // Método público
};

class figura {
  private:
    int x, y;      // Atributos privados
    int color (); // Método privado
  public:
    void dibujar (); // Método público
};
```

Los especificadores de acceso se pueden alternar tantas veces como se quiera:

```
class ejemplo {
  public:
    int a;
  private:
    int b;
  public:
    int c;
  private:
    int d;
};
```

Aunque C++ permite hacer semejante cosa, no es recomendable en absoluto. Lo más lógico sería agruparlos para evitar problemas de legibilidad y facilitar el mantenimiento de los programas:

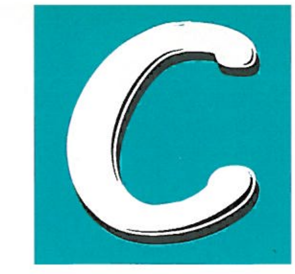
```
class ejemplo {
  private:
    int b, d;
  public:
    int a, c;
};
```

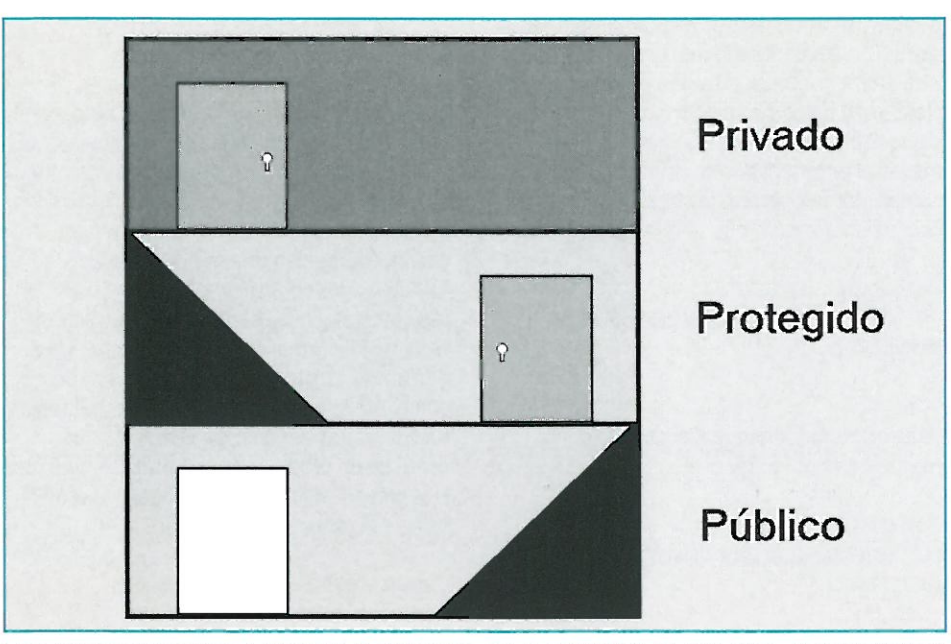
De igual modo, se recomienda no omitir la palabra clave *private* con el fin de evitar pequeños despistes y aumentar la claridad del código fuente.

OMISION DE ATRIBUTOS Y METODOS

En la declaración de una clase se pueden omitir tanto los atributos como los métodos. A primera vista esto puede resultar un tanto extraño. Una clase es realmente práctica si contiene unos datos a los que sólo se puede acceder mediante una serie de métodos perfectamente probados y que garantizan la consistencia de dichos datos. Pues bien, aunque no lo parezca, puede resultar útil disponer de clases sin métodos y clases sin atributos.
Una clase sin métodos equivale a una *struct* de C, siempre que se utilice el especificador *public*. Estas dos definiciones serían equivalentes:

```
struct grupo {
  <lista_datos>
```





LOS TRES TIPOS DE ACCESO A MIEMBROS SE PUEDEN VER COMO UN EDIFICIO DE TRES PISOS, CON DOS PUERTAS CERRADAS CON DISTINTA LLAVE Y OTRA SIEMPRE ABIERTA,

```
};
class grupo {
  public:
    <lista_datos>
};
```

La utilidad de una clase sin funcionalidad es agrupar una serie de datos para ser utilizados en un programa. Por ejemplo, para crear los elementos básicos de un array o los nodos de una estructura de árbol binario.
En contrapartida, la utilidad de una clase sin datos es más que dudosa. Podría ser útil en algún caso muy especial como la generación de números aleatorios:

```
class aleatorio {
  public:
    int obtener ();
};
```

De todos modos, es innegable que esta sería una clase para generar números aleatorios un tanto limitada. Lo normal sería definir una clase que contemplara una semilla para la generación de números:

```
class aleatorio {
  private:
    int semilla;
  public:
    int obtener ();
    void inicializar (int);
};
```

Si se omiten los atributos y los datos se obtiene la clase más sencilla posible. Es la siguiente:

```
class <nombre_clase> {
};
```

Se trata de una clase vacía, es decir, sin miembros. No se han definido datos ni funciones, por lo que sería una clase sin contenido alguno. De todos modos, cualquier compilador de C++ permitirá la creación de una clase con estas características.

CREACION Y DESTRUCCION DE OBJETOS

Es necesario parar la exposición sobre las clases para introducir la creación y la destrucción de objetos. Para crear una instancia de una clase basta con utilizar la sintaxis utilizada con cualquier tipo de datos predefinido:

```
<nombre_clase> <nombre_objeto>;
```

Por ejemplo:

```
class clase {
  /* Contenido de la clase definido por el
programador */
};

int main () {
  /* Sentencias */
  clase objeto;
  /* Más sentencias */
}
```

Aunque el mecanismo de creación de objetos parece sencillo es algo más complicado. Toda clase tiene, al menos, un constructor. Se trata de una función (o varias) que se utiliza para inicializar un objeto de esa clase. Cuando se crea un objeto se llama de forma implícita al constructor correspondiente. Este proceso es transparente al programador, es decir, no se debe preocupar de llamar al constructor porque se hace de forma automática. Sin embargo, este proceso debe ser tenido muy en cuenta a la hora de escribir un programa orientado a objetos.
Los constructores se distinguen de los demás métodos por tener el mismo nombre que la clase. Además, si el programador decide no crear un constructor personalizado para la clase, el compilador crea uno por defecto. La clase anterior:

```
class clase {
  /* Contenido de la clase definido por el
programador */
};
```

Sería vista por el compilador de la siguiente forma:

```
class clase {
  /* Contenido de la clase definido por el
programador */
  clase () {};
};
```

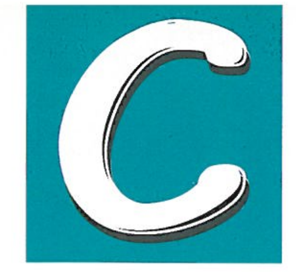
Como se ve en el ejemplo, un constructor no devuelve nada. Por esta razón no es necesario especificar el tipo devuelto, ni tan siquiera *void*. El problema de los constructores por defecto radica en que realizan inicializaciones sencillas. Por ejemplo:

```
class una_clase {
  private:
    int a;
  public:
    /* Miembros públicos */
};
```

El constructor, por defecto, reserva espacio en memoria para un dato de tipo entero. En este caso, dicho comportamiento se puede considerar adecuado. Sin embargo, en este otro:

```
class otra_clase {
  private:
    int a;
    char *cadena;
  public:
    /* Miembros públicos */
};
```

Se reserva espacio para un dato de tipo entero y para un puntero a caracteres pero,



en ningún caso, se reserva espacio para una cadena. Sería necesario definir un constructor que recibiera una cadena de caracteres como parámetro, reservara el espacio necesario en memoria para almacenarla y copiara los contenidos de la misma en el espacio reservado:

```
class otra_clase {
 private:
  int a;
  char *cadena;
 public:
  /* Miembros públicos */
  otra_clase (char *cad) {
   /* Sentencias */
  };
};
```

En este caso, como el programador ha escrito un constructor, el creado por defecto desaparece y no se podría inicializar un objeto de dicha clase con el siguiente código:

```
otra_clase obj;
```

Se produciría un error de compilación por no existir el constructor adecuado. Este otro código sí sería aceptado:

```
char c[] = "Prueba";
otra_clase obj (c);
```

Por supuesto, se podría añadir un nuevo constructor que sirviera para inicializar tanto la cadena como el dato de tipo entero:

```
class otra_clase {
 private:
  int a;
  char *cadena;
 public:
  /* Miembros públicos */
  otra_clase (char *cad) {
   /* Sentencias */
  };
  otra_clase (char *cad, int dato) {
   /* Sentencias */
  };
};
```

Ahora, se podrían crear objetos de la clase de dos formas diferentes:

```
char c1[] = "Prueba 1";
char c2[] = "Prueba 2";
otra_clase obj1 (c1);
otra_clase obj2 (c2, 4);
```

De forma análoga, existe un mecanismo para la destrucción de objetos. Sin embargo, sólo es posible definir un destructor porque sólo debe existir una forma de eliminación del objeto. Se caracteriza por tener el mismo nombre que la clase, pero precedido del símbolo "~" para diferenciarlo de los constructores. Una vez más, si el programador no define uno, el compilador lo crea por defecto. Por ello, es necesario ampliar lo que se dijo antes:

```
class clase {
 /* Contenido de la clase definido por el programador */
};
```

Ante esta situación lo que el compilador realmente ve es lo siguiente:

```
class clase {
 /* Contenido de la clase definido por el programador */
 clase () {};
 ~clase () {};
};
```

Los destructores no admiten parámetros y tampoco devuelven información. Al igual que los constructores son llamados de forma implícita pero, esta vez, en el momento en que deja de utilizarse el objeto. Por ejemplo:

```
void una_funcion () {
 clase objeto;
 /* Sentencias */
}
```

El destructor "*clase::~clase*" sería llamado de forma automática para el objeto creado, al finalizar la ejecución de la función. La razón de esto es muy simple. El objeto sólo es visible dentro del cuerpo de la función y, por tanto, fuera de ella no debe existir.
La utilidad de los destructores creados por el usuario es evidente. Imagínese la clase anterior en la que se crea una cadena de caracteres. El destructor por defecto se limita a liberar tanto el dato entero como el puntero a la clase, pero no se preocupa de liberar la porción de memoria que contiene la cadena. Esto es bastante lógico porque el compilador no puede saber si durante la ejecución del programa la cadena ha sido creada o liberada. Es el programador el que debe definir el comportamiento más adecuado.
Tanto los destructores como los constructores deben estar situados en la sección pública de la clase. De otro modo, no se podrían crear ni destruir objetos:

```
class uno {
 private:
  uno () {};
 public:
  /* Miembros públicos */
};
```

Esta clase compilaría sin problemas pero, en el momento que se intentara crear un objeto, se produciría un error porque el constructor no es accesible.
A la hora de programar hay que tener en cuenta el orden en que se producen las llamadas a los constructores y a los destructores. En la práctica, los constructores se llaman en orden inverso, es decir, se llama primero al constructor más interno y se acaba con el más externo, pero con los destructores sucede exactamente lo contrario. El listado 1 contiene un programa que ilustra este proceso. El resultado por pantalla obtenido al ejecutar dicho programa es el siguiente:

```
Constructor clase tres
Constructor clase dos
Constructor clase uno
Destructor clase uno
Destructor clase dos
Destructor clase tres
```

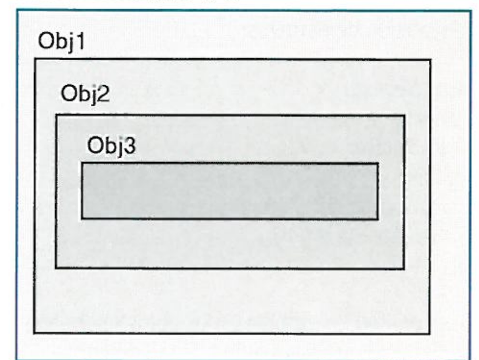
También es posible crear y destruir objetos de forma explícita mediante los operadores *new* y *delete*, que serán tratados en otro momento.

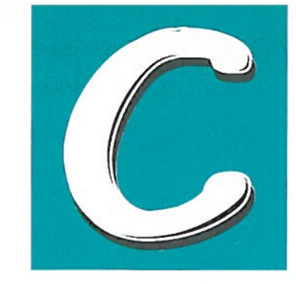
UTILIZACION DE MIEMBROS

Supóngase una clase que tiene un método denominado "definir" que no necesita parámetros. Para llamar a dicho método se utiliza el mismo sistema utilizado con las *struct* de C:

```
class clase {
 private:
  /* Miembros privados */
 public:
  void definir () {
   /* Sentencias */
  };
};
```

EN EL EJEMPLO DEL LISTADO 1, EL OBJETO CREA UN OBJETO DE OTRO TIPO, Y ESTE ULTIMO CREA UN TERCERO, TAMBIEN DE DISTINTO TIPO.






```
int main () {
  /* Sentencias */
  clase objeto;
  objeto.definir ();
  /* Más sentencias */
}
```

Para acceder a los atributos de la clase también se utiliza este mecanismo.

DEFINICION DE LOS CUERPOS

El lector habrá observado que, hasta ahora, no se han puesto más que prototipos de funciones en el interior de las clases. Es evidente que habrá que especificar los cuerpos de las funciones miembros en alguna parte. Existen dos opciones:

1) Definición fuera del cuerpo de la clase:
Retomando el ejemplo de la clase figura, la función dibujar se podría definir (no confundir con declarar) de la siguiente forma:

```
class figura {
  private:
    int x, y;
    int color ();
  public:
    void dibujar (); // Declaración
};

void figura::dibujar () { // Definición
  /* Cuerpo */
}
```

Dentro del cuerpo de la clase se ha declarado la función, mientras que fuera se ha definido, es decir, se ha establecido su contenido. Para ello, ha sido necesario utilizar el operador de resolución de alcance ("::"). La porción de código "*void figura::dibujar ()*" significa que se está haciendo referencia a la función dibujar de la clase figura que devuelve *void* y no necesita parámetros. Si no se hubiera utilizado dicho operador, se habría definido una función global que nada tiene que ver con la clase figura:

```
void dibujar () {
  /* Cuerpo de la función global dibujar */
}
```

De hecho, se pueden poner ambas funciones en un mismo programa fuente y el compilador no arrojará ningún error de redefinición porque afectan a distintos ámbitos.

2) Definición en el propio cuerpo de la clase:
En este caso, por razones obvias, no es necesario utilizar el operador "::" para hacer referencia a la propia clase:

```
class figura {
  private:
    int x, y;
    int color ();
  public:
    void dibujar () {
      /* Cuerpo de la función dibujar */
    };
};
```

Aunque, aparentemente, ambas formas de definición de cuerpo son equivalentes, no es así. En la primera, toda utilización del método dibujar de un objeto figura implica una llamada a la función. En la segunda, sin embargo, se sustituye la llamada por el cuerpo de la función en tiempo de compilación (cuando se compila el programa). Por ejemplo, en el segundo caso este código:

```
int main () {
  figura a;
  a.dibujar ();
  /* Otras sentencias */
}
```

se transformaría en el siguiente:

```
int main () {
  figura a;
  /* Cuerpo de la función dibujar */
  /* Otras sentencias */
}
```

Cuando se utiliza la definición dentro del cuerpo de una clase se dice que la función es "en línea" (*inline*). Es ideal para funciones con muy pocas sentencias que se ejecutan con mucha frecuencia, especialmente dentro de bucles. Con ello se obtiene un ejecutable algo más grande, pero más eficiente porque se evita el mecanismo de llamada y retorno en las funciones de uso frecuente. Por supuesto, no se debe utilizar con funciones cuyo cuerpo sea grande y contenga llamadas a otras funciones.
Existe la posibilidad de crear funciones en línea fuera del cuerpo de una clase. Para ello se utiliza la palabra clave *inline*:

```
inline void figura::dibujar () {
  /* Sentencias de la función dibujar */
}
```

El uso de las funciones *inline* no está restringido a las funciones miembro de una clase. También se puede utilizar para las funciones globales:

```
inline void dibujar () {
  /* Sentencias de la función global dibujar */
}
```

LISTADO 1. EJEMPLO DE CREACION Y DESTRUCCION DE OBJETOS

```
#include <iostream.h>

class tres {
  public:
    tres () {
      cout << "Constructor clase tres\n";
    };
    ~tres () {
      cout << "Destructor clase tres\n";
    };
};

class dos {
  private:
    tres obj3;
  public:
    dos () {
      cout << "Constructor clase dos\n";
    };
    ~dos () {
      cout << "Destructor clase dos\n";
    };
};

class uno {
  private:
    dos obj2;
  public:
    uno () {
      cout << "Constructor clase uno\n";
    };
    ~uno () {
      cout << "Destructor clase uno\n";
    };
};

int main () {
  uno obj1;
  return 0;
}
```

Una última observación respecto a las funciones en línea. La definición de una función en línea debe situarse en el mismo archivo en el que ha sido declarada.

Próxima entrega

En el próximo número se verán, entre otras cosas, los miembros estáticos, los miembros constantes, las funciones amigas y el puntero *this*.
Si surgen dudas sobre temas relacionados con el presente curso contactar con el autor: cursocpp@usa.net



Autor: José C. Remiro

Los grafos

La estructura de datos dinámica nò lineal más genérica que existe es el grafo. Esta estructura tiene un gran número de aplicaciones, entre las que se encuentran la planificación de procesos y el análisis de redes. En este artículo se presentan varias formas de implementar un grafo junto con algoritmos que hacen uso de ellos.

Un grafo se compone de un conjunto de vértices y arcos, cada uno de los cuales une a dos vértices del anterior conjunto. En general, los vértices contendrán información referente a ellos, mientras que la información contenida en los arcos establece dependencias entre los vértices a los que relaciona directamente. Un arco queda determinado por los vértices a los que une; si el orden en que se representa un arco es relevante entonces el grafo se denomina grafo dirigido, en otro caso, se denominará no dirigido. Por ejemplo, suponiendo que se disponga de un grafo con nodos numerados, el par (1,3) denotará el arco que une a los nodos etiquetados por 1 y por 3. En un grafo dirigido, el anterior arco sería diferente del arco (3,1), mientras que en un grafo no dirigido dicho arco sería el mismo. Según las anteriores definiciones entre dos vértices en un grafo dirigido puede haber, a lo sumo, 2 arcos, mientras que en uno no dirigido puede haber como mucho 1 arco. El número máximo de arcos que puede contener un grafo no dirigido es de $n(n-1)/2$, donde n es el número de vértices del grafo; en un grafo dirigido el número máximo de arcos es de $n(n-1)$. Si un grafo dispone de todos los posible arcos entre sus vértices entonces se denomina completo. En la figura 1 se observa una serie de ejemplos de grafos. El grafo B es, en particular, un árbol (se trata de un grafo no dirigido, acíclico y conexo). El grafo C es un ejemplo de grafo completo pues dispone de todos los posibles arcos entre sus nodos. Por su parte, el grafo A es un grafo dirigido, la flecha indica el orden en el que se establecerán los arcos. Como se puede observar también en la figura, el conjunto de arcos no es más que una lista de pares de vértices.

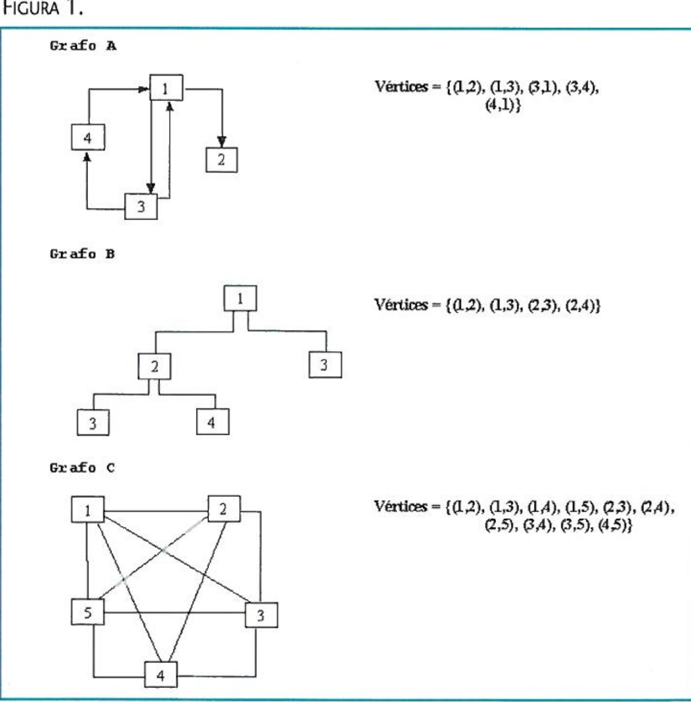
Dos vértices que son extremos de un arco se denominan adyacentes, mientras que el arco con respecto a dichos vértices se denomina incidente. Un sub-grafo de un grafo no es más que un subconjunto de nodos y vértices de un grafo. Se denomina camino entre dos vértices v_r y v_s a un conjunto ordenado de arcos de la forma $(v_r,v_{k+1}),...,(v_{k+n},v_s)$. Un camino simple es un camino en el que todos los vértices contenidos en sus arcos son distintos salvo, posiblemente, los vértices inicial y final. Un camino cuyo primer arco comienza con el mismo vértice con el que termina el arco final se denomina bucle, camino cíclico o simplemente ciclo. Se define el grado de un vértice como el número de arcos que inciden sobre él. Se dice que un grafo es conexo si desde cualquier vértice existe un camino a cualquier otro de los vértices que lo compone.

REPRESENTACIONES

Existen varias representaciones posibles de los grafos, aparte de las que se expondrán en este apartado, éstas son las más comunes. La selección por parte del programador de una representación en particular dependerá de las operaciones que desee realizar sobre el grafo.

Supuesto que el grafo posea n vértices es posible representarlo utilizando una matriz de n filas y n columnas cuyo tipo elemental de datos, en principio, puede ser lógico (boolean). Si existe el arco que une los vértices (v_i,v_j), entonces la posición (i,j) de la matriz tendrá el valor **true**, en cualquier otro caso será **false**. A dicha matriz se la denomina matriz de adyacencia. Si en una de tales matrices se representa un grafo no dirigido, entonces la matriz resultante será simétrica respecto de la diagonal; esta propiedad no es cierta para los grafos dirigidos. Las principales ventajas ofrecidas por la matriz de adyacencia son que se puede determinar inmediatamente si existe un arco entre dos vértices accediendo al elemento correspondiente de la matriz, y es posible determinar el grado de cualquier vértice (número de arcos que inciden en un vértice) sumando 1 a un acumulador cada vez que en la fila correspondiente al número de vértice se encuentre un elemento con valor **true** (si se trata de un grafo no dirigido); si el grafo es dirigido, contando los elementos cuyo contenido sea **true** en la fila correspondiente al vértice se obtendrá el número de arcos de salida, mientras que si se cuentan los de la columna se obtendrán los arcos de salida desde el vértice. La principal desventaja de la matriz de adyacencia es la cantidad de elementos que quedan desaprovechados si el grafo a representar dispone de gran cantidad de vértices y escasos arcos, pues los algoritmos que tengan que inspeccionar la

FIGURA 1.





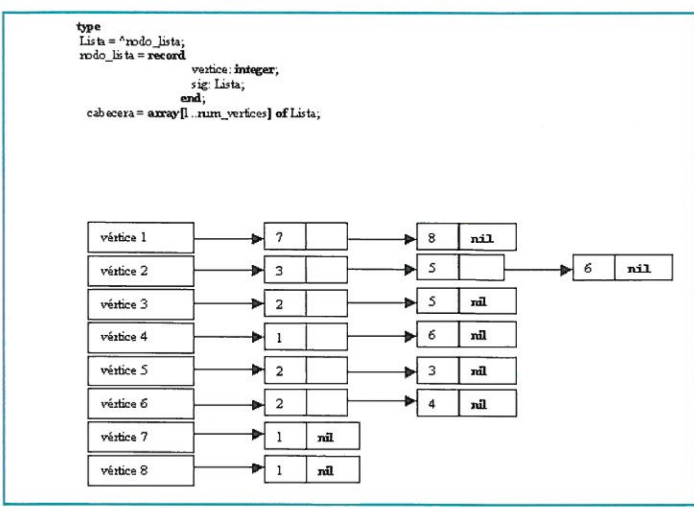
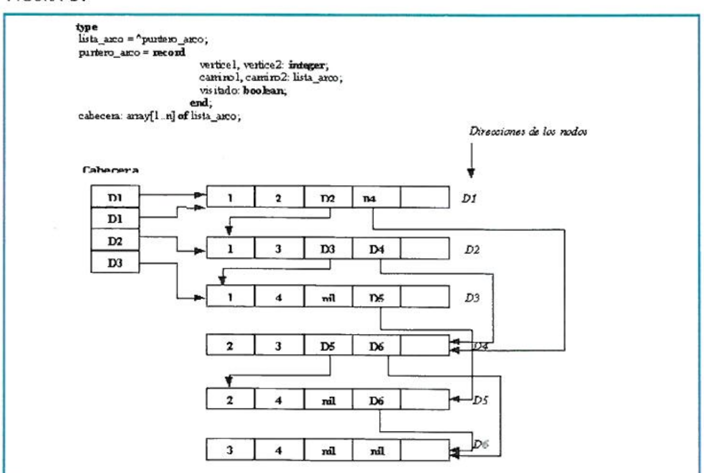


FIGURA 2.

totalidad de la matriz deberán acceder a n^2-n elementos, siendo n el número de vértices del grafo. Otra posibilidad para representar un grafo consiste en utilizar las listas de adyacencia. En éstas se dispone de una lista por cada uno de los vértices del grafo; cada uno de los nodos de dicha lista representan a los vértices que son adyacentes al vértice de la cabecera. Para facilitar el acceso secuencial a la lista de cada uno de los nodos se dispone de un array de punteros y cada elemento de éste apunta a cada una de las listas de los vértices, cada uno de los nodos de las listas contendrá el número de vértice. En la figura 2 se encuentra la estructura de datos correspondiente a la lista de adyacencia, junto con un ejemplo. Si se desea conocer el grado de un vértice en un grafo no dirigido, basta con contar el número de elementos de la lista correspondiente a dicho vértice; para conocer el número total de arcos es necesario recorrer todas las listas. En un grafo dirigido el número de arcos de entrada de un vértice se puede conocer contando el número de nodos de su lista de adyacencia, sin embargo, será necesaria una inspección de todos los nodos de las listas para contar el número de arcos de salida. Si es necesario calcular este dato con cierta frecuencia, para facilitar su cálculo, es posible implementar una lista similar a la anterior pero inversa denominada lista de adyacencia inversa.

Las multi-listas de adyacencia son otra estructura de datos que permiten representar un grafo; éste tipo de representación evita tener dos entradas para un mismo arco como ocurre en el caso de las listas de adyacencia. En la figura 3 se puede observar que cada nodo almacena un par de vértices correspondiente a cada uno de los arcos del nodo, además, dispone de dos punteros que permiten seguir el rastro de los arcos que contienen como origen o destino uno de los vértices contenidos en el nodo y un campo de tipo lógico para apoyar los procesos de recorrido del grafo. Aunque no se ha mencionado hasta el momento, en multitud de ocasiones los arcos de un grafo pueden contener una serie de pesos o costes asociados al arco; la representación con multi-listas permite almacenar en cada nodo esta información sin más que añadir un nuevo campo en la estructura correspondiente a un nodo.

FIGURA 3.




```
procedure profundidad (num_ver: integer);
 var
  aux_ver: integer;
 begin

   {visitado es un array de tantas componentes
    como vértices tenga el grafo, de tipo boolean,
    inicializadas a false. Indican si el nodo ha
    sido visitado o no con anterioridad. Se trata de
    una variable global}

  visitado[num_ver]:= true;
  for cada vértice aux_ver adyacente a num_ver

     {esta instrucción depende del tipo de representación
      del grafo, la más sencilla resulta de seleccionar
      la matriz de adyacencia}

    do
     if not visitado[aux_ver]
       then
        profundidad(aux_ver);
 end;

procedure anchura (num_ver: integer);
 var
  aux_ver: integer;
  c: cola;
 begin
  visitado[num_ver]:= true;
  inicia_cola(c);
  apila (c, num_ver);
  while not pilavacia (c)
       do
        begin
         desapila (c, num_ver);
         for cada vértice aux_ver adyacente a num_ver
             do
              if not visitado[aux_ver]
                then
                 begin
                  apila(c, aux_ver);
                  visitado[aux_ver]:= true;
                 end;
        end;
 end;
```

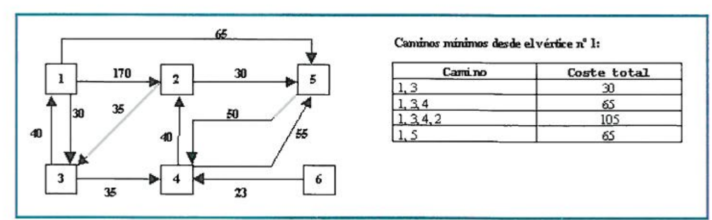
FIGURA 4.

RECORRIDO DE UN GRAFO

Uno de los procesos más habituales que se realiza en un grafo es determinar los vértices que pueden ser visitados desde un vértice dado; puesto que un grafo es una estructura de datos no lineal habrá que establecer un orden para visitar los vértices, así se puede hablar de búsqueda en profundidad o en anchura dependiendo del orden en que sean visitados los vértices. Una vez más, la recursión será de gran ayuda a la hora de diseñar los algoritmos.

Una vez seleccionado el vértice, desde donde se iniciará la búsqueda en profundidad, tomará un vértice adyacente al anterior que aún no haya sido visitado; este proceso se repetirá con uno de los vértices adyacentes del último vértice visitado. Cuando todos los vértices adyacentes de un vértice hayan sido visitados (se ha llegado al final del camino) se procede a seleccionar el siguiente vértice adyacente al vértice anterior, proceso que se repetirá hasta que se visiten todos los vértices adyacentes del vértice inicial. De todas formas el algoritmo de la figura 4 muestra con más sencillez el proceso.

La búsqueda en anchura desde un vértice dado comienza visitando todos y cada uno de los vértices adyacentes al inicial; posteriormente, se procede a realizar los vértices adyacentes de los vértices anteriormente visitados. La figura 4 contiene la estructura del algoritmo de búsqueda en anchura utilizando una cola como estructura



Caminos mínimos desde el vértice nº 1:

Camino	Coste total
1, 3	30
1, 3, 4	65
1, 3, 4, 2	105
1, 5	65

FIGURA 5.

de apoyo para el recorrido (este tipo de estructuras fueron descritas en anteriores artículos).
A partir de los algoritmos de búsqueda en anchura y profundidad se puede construir un algoritmo que decida si dos vértices dados están conectados. Un algoritmo más elaborado consistiría en aquel que detecta todos los componentes conectados del grafo; para construir este algoritmo basta aplicar para cada uno de los vértices del grafo cualquiera de los anteriores algoritmos de búsqueda, anotando cada vértice visitado durante dicho proceso.

CAMINO MINIMO

Los grafos pueden representar multitud de situaciones reales. Imagínese un conjunto de ciudades comunicadas entre sí por carreteras, además, cada carretera tiene asociada un tiempo medio para llegar de una ciudad a otra. Se podría representar esta situación mediante un grafo en el que los vértices representan las ciudades, mientras que la existencia de una carretera entre dos de aquellas estaría representada por un arco del grafo; además, cada arco tiene asociado un coste o peso representado por el tiempo medio en recorrer el camino que separa una de otra. Dos preguntas habituales que se pueden plantear ante esta situación son: ¿Existe un camino entre dos ciudades?, de existir más de un camino ¿cuál es el más corto en relación a los costes de tiempo asignados? La figura 5 muestra un grafo de este tipo.
El primer algoritmo que se analizará será aquel que tiene fijado el vértice de origen y proporciona los caminos con coste mínimo para cada uno de los vértices a los que se puede llegar a partir de él. Supóngase que se dispone de un conjunto de vértices, al que se denominará M, incluido el inicial, que dispone de los caminos mínimos ya calculados (este conjunto inicialmente tendrá como único elemento el vértice desde el que se inician todos los caminos), además se supondrá que los caminos se generarán en orden creciente respecto a su correspondiente coste en tiempo.
La primera consideración a realizar consiste en que al generar el siguiente camino desde el vértice inicial a otro vértice que no está incluido en el conjunto M se deberán utilizar vértices que ya se encuentran en dicho conjunto, o bien, se utilizará el camino directo desde el vértice inicial a éste. Esto es así, debido a que si existiese un camino con vértices no pertenecientes al conjunto M éstos serían de menor longitud que la calculada, por tanto, si se están generando los caminos en orden creciente respecto a su coste en tiempo dicho camino debería estar en M, por lo tanto se llegaría a una contradicción de ser falsa la afirmación inicial. Esta consideración establece el orden en el que se generarán los caminos, a la vez que restringe el número de arcos que deberán ser consultados para realizar el cálculo del siguiente camino mínimo. La segunda consideración a realizar consiste en que el siguiente vértice a incluir en el conjunto M sería aquel que, no estando en M, tuviese el camino de mínimo coste con origen en uno de los vértices del conjunto M. Esta consideración establece cómo aumentar el conjunto M, a la vez que se restringe el cálculo de los costes de los caminos.
En la figura 6 se puede encontrar el algoritmo para calcular los caminos con mínimo coste desde un vértice inicial, en él se asume que los vértices están representados por números enteros comenzando por 1, siendo **max_vert** el número de vértices de que dispone el grafo. El array **seleccion** dispone de tantas posiciones como vértices tiene el grafo y su tipo de datos base es *boolean*; permite representar el conjunto de vértices seleccionados a medida que se construyen los caminos, es decir, en cada momento las componentes con valor **true** indicarán que dicho vértice pertenece al conjunto M descrito en el anterior párrafo. La matriz de adyacencia puede ser sustituida por una matriz de costes, **mat_costes**, en la que cada componente conectada por un camino directo tiene almacenado el coste para dicho camino; cuando no hay un camino directo entre dos vértices la correspondiente posición almacenará el valor **maxint** supuesto que los costes estén representados por un valor de tipo entero. La matriz **distancias** contiene, como su nombre indica, las correspondientes distancias desde el vértice inicial a cada uno de los vértices. Los valores contenidos en esta matriz serán modificados a medida que se realice el cálculo del camino mínimo.
El algoritmo comienza inicializando las distancias con los costes correspodientes a la matriz de costes y a cada uno de los elementos del array **seleccion** se les asigna el valor **false** para indicar que aún no se ha inspeccionado ningún camino. A continuación, el primer elemento del array **seleccion** pasa a ser el vértice inicial, que aparece como parámetro del procedimiento; evidentemente su distancia será establecida a 0. La instrucción **for** más externa se encarga de determinar los caminos mínimos (son en número **max_vert-1**), seleccionando en la variable **aux_vert_1** el vértice que aún no habiendo sido seleccionado (**selección[aux_vert_1]** será igual a **false**) tiene la mínima distancia en el array **distancias;** este proceso es el realizado por la función **elegir_minima.** Una vez encontrado, se establece dicho vértice como seleccionado, a continuación se procede a comprobar si entre los vértices aún no seleccionados existe alguno que conectado con el anterior tenga menor distancia que la incluida en la matriz de distancias y, si se encuentra alguno, entonces se establece su correspondiente distancia a la calculada.
Una extensión natural del anterior algoritmo consiste en calcular el camino mínimo entre todos los vértices que se encuentran en el grafo; bastaría con llamar al procedimiento variando el vértice que se encuentra como primer parámetro. Sin embargo, existe otro método para calcular todos los caminos mínimos de todos los vértices aplicando programación dinámica; a continuación se describirá. De manera similar al anterior algoritmo se dispondrá en la matriz de adyacencia de los tiempos asociados a cada uno de los arcos

FIGURA 6.

```
type
 matriz_ady= array [1..max_vert, 1..max_vert] of integer;
 vector_dst= array[1..max_vert] of integer;
 vector_sel= array[1..max_vert] of boolean;
...
....

procedure camino_minimo (v_inicial: integer; mat_costes: matriz_ady;
                         var distancias: vector_dst; n: integer);
var
 seleccion: vector_sel;
 indice: integer;
 vert_aux_1, vert_aux_2: integer;
 coste_aux: integer;
begin
 for indice:= 1 to max_vert
    do
     begin
      seleccion[indice]:= false;
      distancias[indice]:= mat_costes[v_inicial, indice];
     end;
 seleccion[v_inicial]:= true;
 distancias[v_inicial]:= 0;
 for indice:= 1 to max_vert -2
    do
     begin
      vert_aux_1:= elegir_minima(distancias, max_vert);
      seleccion[vert_aux_1]:= true;
      for vert_aux_2:= 1 to max_vert
         do
          if not seleccion[vert_aux_2]
            then
             begin
              coste_aux:= distancias[vert_aux_1]+ mat_costes [vert_aux1, vert_aux_2]
              if coste_aux < distancias[vert_aux_2]
                then
                 distancias[vert_aux_2]:= coste_aux;
             end;
     end;
end;
```

```
procedure total_min ( costes: matriz_ady; var mat_min: matriz_ady;
                      num_ver: integer);
var
 i, j, k: integer;
 aux_cost: integer;
begin
 for i:=1 to num_ver
    do
      for j:= 1 to n
        do
          mat_min[i,j]:= costes[i,j];
 for k:= 1 to num_ver
    do
      for i:=1 to num_ver
        do
          for j:= 1 to num_ver
            do
              begin
               aux_cost:= mat_min[i,k] + mat_min[k,j];
               if aux_cost < mat_min[i,j]
                 then
                   mat_min[i,j]:= aux_cost
              end;
end;
```

FIGURA 7.

presentes en el grafo; la diagonal de esta matriz estará constituida por ceros (el coste de ir de un vértice así mismo es 0), mientras que los arcos que no estén presentes en la matriz tendrán el valor **maxint**. Se denotará con C^k a la matriz que contiene los costes mínimos para ir de un vértice a otro sin utilizar un vértice cuyo índice sea mayor que *k*. Teniendo en cuenta esta definición, si se calcula C^n se habrán obtenido los caminos mínimos de todos y cada uno de los vértices, pues en ella no hay restricciones sobre la utilización de los vértices en los caminos. Falta ahora establecer cómo a partir de la matriz C^{k-1} puede obtenerse la matriz C^k, ya que se conoce la matriz inicial.

Para realizar el cálculo de la matriz de orden *k* a partir de la de orden *k-1* basta con identificar los casos que pueden darse, así:

Si al camino mínimo entre dos vértices *i* y *j* no pertenece el vértice de índice *k*, entonces las componentes de posición *(i,j)* en ambas matrices coinciden.

Si el camino mínimo entre dos vértices *i* y *j* pasa por el vértice *k*, entonces los costes de los caminos que van de *i* a *k* y de *k* a *j* son los que se encuentran en la matriz de índice *k-1*, siendo el valor de la posición *(i,j)* de la matriz de índice *k* la suma de los costes de las posiciones de la matriz de índice *k-1*, *(i,k)* y *(k,j)*.

Por tanto, de los anteriores puntos se obtiene que para calcular la matriz de costes de índice *k*, a partir de la matriz de costes de índice *k-1*, habrá que seleccionar el valor mínimo de entre la posición *(i,j)* de la matriz de índice *k-1* y la suma de las posiciones *(i,k)* y *(k,j)* de dicha matriz.

En la figura 7 se encuentra el algoritmo que realiza los anteriores cálculos, dispone como parámetros de la matriz de costes, de la matriz de costes mínimos y del número de vértices que contiene el grafo. Comienza este algoritmo inicializando la matriz de costes mínimos con la matriz de costes; posteriormente, en cada iteración de la instrucción **for** más externa, se calculan cada una de las matrices de costes del índice que corresponda.

ORDEN TOPOLOGICO

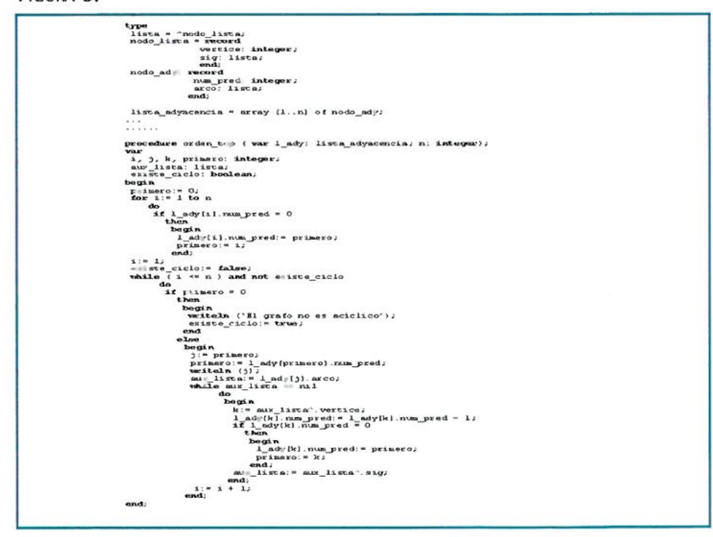
Los grafos también permiten representar una serie de tareas que se deben realizar en un orden determinado; por ejemplo, un proyecto informático puede ser dividido en una serie de fases de tal forma que una no pueda comenzar hasta que otra haya finalizado, a su vez cada fase está dividida en tareas que pueden ser realizadas de forma paralela, o bien, para comenzar una o varias de ellas es necesario que finalicen otras; así, cada vértice del grafo representaría una tarea mientras que los arcos representarían las dependencias temporales entre ellas. A este tipo de grafos se les conoce por el nombre de redes. En una red se dice que un vértice *i* es predecesor de otro *j* si existe un camino desde *i* hasta *j* (j es sucesor de i); si dicho camino es un arco se dice que el vértice *i* es un predecesor inmediato del vértice *j*. En una red se podrá establecer un orden parcial entre sus elementos si las relaciones que se establecen entre ellos son irreflexivas (no hay arcos que se inician y terminan en el mismo vértice) y transitiva (si *i* es predecesor de *j*, y *j* es predecesor de *k*; entonces *i* es predecesor de *k*). En definitiva, estas restricciones establecen que en el grafo no hay ciclos (grafo acíclico).

La idea para realizar un algoritmo que establezca si un grafo no tiene ciclos y, en tal caso, establecer un orden entre sus vértices es bastante sencilla, pues basta con detectar aquellos vértices que no tienen predecesores (ningún arco incide sobre ellos); una vez determinados se eliminan junto con los arcos que tienen su origen en ellos, el proceso se repite hasta que el grafo no dispone de ningún vértice (se ha establecido un orden) o hasta que todos los vértices del grafo tienen predecesor, en cuyo caso el grafo no es acíclico y, por tanto, no es posible establecer un orden entre sus vértices.

Básicamente el algoritmo deberá realizar las siguientes funciones: decidir si un vértice dispone de predecesores y eliminar todos los arcos cuyo origen sea un vértice determinado. Para realizar la primera tarea basta con incluir dentro de cada nodo que representa a un vértice en la cabecera de la lista de adyacencia un nuevo campo que contenga el número de predecesores del vértice, mientras que para la segunda tarea será suficiente con seleccionar como representación del grafo la lista de adyacencia (ver el apartado "representaciones").

En la figura 8 puede verse la implementación del algoritmo para establecer un orden topológico en un grafo; el array de cabeceras ha sido dotado de un nuevo campo que realiza dos funciones: antes de iniciarse el algoritmo contiene el número de predecesores que tiene el vértice correspondiente, cuando se vuelve cero y puesto que no se usará posteriormente, se utiliza para crear una pila compuesta de los vértices que se quedan sin predecesores y cuya cima es la variable **primero**. La primera instrucción **for** está dedicada a crear la pila con los vértices que inicialmente no disponen de ningún predecesor. La instrucción **while** más externa finalizará cuando se han visitado los *n* vértices (se ha terminado de ordenar) o cuando la pila ha quedado vacía (no hay vértices sin predecesores y, por lo tanto, existe un bucle). La instrucción **while** más interna está dedicada a recorrer la lista correspondiente a uno de los vértices que se quedó sin predecesores; para cada uno de los vértices de dicha lista se reduce en uno su número de predecesores, pasando a ocupar un lugar en la pila si se ha quedado sin ninguno. Sin duda, el algoritmo parece oscuro por la doble función que realiza el campo **num_pred**, sin embargo, gracias a esta doble función no es necesaria la inclusión de una estructura de datos extra para albergar a los nodos eliminados.

FIGURA 8.



```
type
 lista = ^nodo_lista;
 nodo_lista = record
                vertice: integer;
                sig: lista;
                end;
 nodo_ady = record
              num_pred: integer;
              arco: lista;
             end;

 lista_adyacencia = array [1..n] of nodo_ady;
...
......

procedure orden_top ( var l_ady: lista_adyacencia; n: integer);
var
 i, j, k, primero: integer;
 aux_lista: lista;
 existe_ciclo: boolean;
begin
 primero:= 0;
 for i:= 1 to n
     do
      if l_ady[i].num_pred = 0
        then
         begin
          l_ady[i].num_pred:= primero;
          primero:= i;
         end;
 i:= 1;
 existe_ciclo:= false;
 while ( i <= n ) and not existe_ciclo
      do
       if primero = 0
         then
          begin
           writeln ('El grafo no es aciclico');
           existe_ciclo:= true;
          end
         else
          begin
           j:= primero;
           primero:= l_ady[primero].num_pred;
           writeln (j);
           aux_lista:= l_ady[j].arco;
           while aux_lista <> nil
                do
                 begin
                  k:= aux_lista^.vertice;
                  l_ady[k].num_pred:= l_ady[k].num_pred - 1;
                  if l_ady[k].num_pred = 0
                    then
                     begin
                      l_ady[k].num_pred:= primero;
                      primero:= k;
                     end;
                  aux_lista:= aux_lista^.sig;
                 end;
            i:= i + 1;
          end;
end;
```

Mapeado de Texturas

El mapeado de texturas plantea nuevas dificultades a la hora de cortar un polígono con los bordes de la pantalla. En el presente capítulo se verá el modo de resolverlas.

Pero el análisis de los problemas del recorte de polígonos al usar mapeado de texturas no será lo único que se tratará en este número. En la entrega anterior se realizó la rutina que se encarga de pintar un polígono con una textura, dejándose a los lectores la tarea de descifrar su funcionamiento. Hoy se explicará, paso a paso, cómo funciona dicha rutina, ya que su correcto entendimiento permitirá posteriores mejoras y optimizaciones.
En el ejemplo también se realiza el corte en la coordenada Z del Z-buffer, ya que para una correcta impresión de los polígonos es necesario calcular también este valor. El sistema utilizado es el mismo que en el mapeado de texturas, sólo que es algo más simple al tratarse de un sólo valor (Z) en lugar de dos (X e Y de mapping).

EL PROBLEMA DEL MAPPING

El recorte de polígonos elimina la parte de un polígono que queda fuera de la pantalla, generando un nuevo polígono recortado que queda totalmente dentro del área de visualización. Al estar usándose texturas será necesario calcular unas nuevas coordenadas de mapping para el polígono. El proceso a realizar se entenderá con un simple ejemplo: Supongamos que se está dibujando un cuadrado que está completamente recto, apareciendo justo la mitad del cuadrado fuera de la pantalla y la otra parte dentro. Si un vértice que está dentro tiene la coordenada "X" de mapeado de texturas, con un valor igual a 0, y otro vértice que está fuera tiene la misma coordenada "X" igual a 100, y ambos están unidos formando uno de los lados del cuadrado, resulta que al realizar su clipping el vértice que aparece a un lado de la pantalla y que sustituye al que estaba fuera tiene un valor en su coordenada "X" de mapeado igual a 50, es decir, si un vértice tenía el valor 0 y el otro tiene 100 lo que se ha hecho es calcular el valor que corresponde en el punto de corte con el extremo de la pantalla, que al ser justo la mitad del lado del cuadrado es igual a 50.
Al realizar el corte de un polígono será necesario almacenar los nuevos valores de mapeado de texturas sin perder los originales, ya que los valores iniciales son siempre necesarios para calcular el mapeado en posteriores impresiones del polígono. Debido a eso se ha ampliado la estructura de los vértices con dos nuevos valores. Ahora las variables en coma flotante "*mapx*" y "*mapy*" contienen las coordenadas de mapping del vértice originales, mientras que "*mapx2*" y "*mapy2*" contienen las resultantes tras pasar los polígonos por los algoritmos de corte:

```
struct vertex_type
{
 float rx,ry,rz; //coordenadas relativas al punto central de un polígono
 float wx,wy,wz; //coordenadas en el mundo tridimensional, tras rotar
 float sx,sy,zbuff; //coordenas en pantalla
 float mapx,mapy;
 float mapx2,mapy2;
};
```

La función "*PolygonCalcule2D*" ha sido modificada para copiar los valores de mapeado de "*mapx*" a "*mapx2*", y de "*mapy*" a "*mapy2*" en cada ciclo de impresión:

```
void PolygonCalcule2D(polygon_type &pol)
{
 sint n;
for(n=0;n<(pol.n_vertex);n++)
 {
 //Transforma las coordenadas 3D en coordenadas 2D
 (pol.vertex[n].sx)=((pol.vertex[n].wx)*lenx)/(pol.vertex[n].wz)+origenx_3d;
 (pol.vertex[n].sy)=((pol.vertex[n].wy)*leny)/(pol.vertex[n].wz)+origeny_3d;
 //Coordenada Z para el cálculo del Zbuffer
 (pol.vertex[n].zbuff)=pol.vertex[n].wz;
 //Mapeado de Texturas
 (pol.vertex[n].mapx2)=pol.vertex[n].mapx;
 (pol.vertex[n].mapy2)=pol.vertex[n].mapy;
 }
}
```

A continuación viene la función de corte de polígonos con el extremo izquierdo de la pantalla, la cual ha sido mejorada para que soporte mapeado de texturas y Z-buffer. En el presente curso no se incluirá la mejora de los tres restantes cortes 2D y del corte 3D, que se ha hecho así por motivos de espacio. Cualquier lector que entienda el funcionamiento de este sistema de corte podrá implementar fácilmente las mejoras en las restantes funciones de corte, debido a que lo único que cambia son las coordenadas a cortar (ya sean X, Y o Z) y la condición de corte (ya salga el polígono por el límite inferior o por el límite mayor de una coordenada):

```
//Esta función hace el corte en 2D del limite Izquierdo de la pantalla
//
//Tiene como parámetro de entrada un polígono, el cual será modificado.
//Devuelve el número de vértices del polígono recortado
sint Polygon_Clip_2dx1_Updated(polygon_type &pol)
{
 polygon_type pol2;
 //Punteros que recorren los dos polígonos, el del parametro de entrada
 //y el temporal
 vertex_type *ver,*indice2;
 //Indice3 contendrá el número de vértices que tendrá el polígono tras
 //hacer el corte
 sint n,indice3=0;
 indice2=&(pol2.vertex[0]);
 float lastx,lasty,lastzb,dx,dy,dzb;
 //Importante
 //El vértice anterior del primer vértice es el último vértice
 ver=&(pol.vertex[(pol.n_vertex)-1]);
 lastx=(ver->sx);
 lasty=(ver->sy);
```

Aparecen nuevas variables para contener el valor en Z y en las dos coordenadas de mapping del vértice anterior:



```
//Corte para el Z-buffer
lastzb=(ver->zbuff);
//Corte para el mapeado de texturas
float dmx,dmy;
float dzb2,dzb3;
float lastmx=ver->mapx2;
float lastmy=ver->mapy2;
ver=&(pol.vertex[0]);
//Se recorren todos los vértices del polígono
for(n=0;n<(pol.n_vertex);n++)
{
 //Caso 1, el vértice ACTUAL sale por el extremo izquierdo
 //de la pantalla
 if((ver->sx)<minx)
 {
  //Caso 2, el vértice ANTERIOR sale por el extremo izquierdo
  //de la pantalla (y el actual fuera)
  if(lastx>=minx)
  {
   //Cálculo del corte
   dx=(ver->sx)-lastx;
   dy=(ver->sy)-lasty;
   (indice2->sx)=minx;
   (indice2->sy)=lasty-(dy*(lastx-minx)/dx);
```

Al estar usándose texturas será necesario calcular unas nuevas coordenadas de mapping para el polígono

Todas estas operaciones para realizar el cálculo del corte son, en realidad, unas sencillas ecuaciones de primer grado que se basan en una simple regla de tres. Las variables "*dzb*", "*dzb2*" y "*dzb3*", así como "*dmx*" y "*dmy*" almacenan valores intermedios que son la diferencia entre las coordenadas que forman cada uno de los lados del polígono, y se usan para el posterior cálculo de las ecuaciones:

```
   //Corte para el Zbuffer y para el mapeado de texturas
   dzb=(ver->zbuff)-lastzb;
   (indice2->zbuff)=lastzb-(dzb*(lastx-minx)/dx);
   dzb2=(ver->zbuff)+(lastzb);
   dzb3=(indice2->zbuff)+(lastzb);
   dmx=(ver->mapx2)-lastmx;
   dmy=(ver->mapy2)-lastmy;
   (indice2->mapx2)=lastmx-(dmx*dzb3*(lastx-minx)/(dx*dzb2));
   (indice2->mapy2)=lastmy-(dmy*dzb3*(lastx-minx)/(dx*dzb2));
   indice2++;indice3++;
  }
 }
 else
 {
  //Caso 3, el vértice ANTERIOR sale por el extremo izquierdo
  //de la pantalla (y el actual dentro)
  if(lastx<minx)
  {
   //Cálculo el corte
   dx=(ver->sx)-lastx;
   dy=(ver->sy)-lasty;
   (indice2->sx)=minx;
   (indice2->sy)=(ver->sy)-(dy*(ver->sx-minx)/dx);
   //Corte para el Zbuffer y para el mapeado de texturas
   dzb=(ver->zbuff)-lastzb;
   (indice2->zbuff)=(ver->zbuff)-(dzb*(ver->sx-minx)/dx);
   dzb2=(ver->zbuff)+(lastzb);
   dzb3=(indice2->zbuff)+(ver->zbuff);
   dmx=(ver->mapx2)-lastmx;
   dmy=(ver->mapy2)-lastmy;
   (indice2->mapx2)=(ver->mapx)-(dmx*dzb3*(ver->sx-minx)/(dx*dzb2));
   (indice2->mapy2)=(ver->mapy)-(dmy*dzb3*(ver->sx-minx)/(dx*dzb2));
   indice2++;indice3++;
  }
  //Caso 4, el vértice ACTUAL queda dentro
  //de la pantalla y no sufre modificaciones
  //Vértice sin modificar
  (indice2->sx)=(ver->sx);
  (indice2->sy)=(ver->sy);
  //Z-buffer
  (indice2->zbuff)=(ver->zbuff);
  //Mapeado de texturas
  (indice2->mapx2)=(ver->mapx);
  (indice2->mapy2)=(ver->mapy);
  indice2++;indice3++;
 }
```

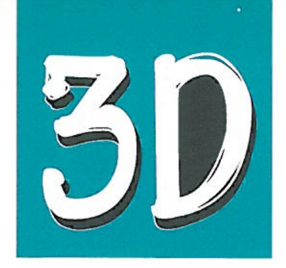
Todas las actualizaciones de los vértices deben de ser ampliadas incluyendo a las nuevas variables que los componen:

```
 //Actualización del vértice anterior
 lastx=(ver->sx);
 lasty=(ver->sy);
 //Z-bbufer
 lastzb=(ver->zbuff);
 //Mapeado de Texturas
 lastmx=(ver->mapx2);
 lastmy=(ver->mapy2);
 ver++;
}
```

Al realizar el corte de un polígono habrá que almacenar los nuevos valores de mapeado de texturas sin perder los originales

Finalmente se hace necesario actualizar también las nuevas coordenadas de mapeado de texturas y de profundidad en el z-buffer:

```
//Actualización del parámetro de entrada "pol"
pol.n_vertex=indice3;
for(n=0;n<indice3;n++)
{
 pol.vertex[n].sx=pol2.vertex[n].sx;
 pol.vertex[n].sy=pol2.vertex[n].sy;
 //Z-buffer
 pol.vertex[n].zbuff=pol2.vertex[n].zbuff;
 //Texturas
 pol.vertex[n].mapx2=pol2.vertex[n].mapx2;
 pol.vertex[n].mapy2=pol2.vertex[n].mapy2;
}
return indice3;
}
```



Ejercicios propuestos

La labor principal del lector será poner a prueba sus conocimientos hasta ser capaz de realizar por sí mismo las mejoras en las otras tres funciones de recorte de polígonos 2D, y en la función de corte en 3D para que puedan funcionar con texturas y Z-buffer.
Se advierte que, aunque es una tarea fácil, puesto que sólo se trata de modificar el recorte por el extremo izquierdo que se ha usado de ejemplo, es muy posible que se cometa algún error en alguna de las múltiples ecuaciones que incorpora la función. Así que el lector no debe desanimarse por los posibles fallos que cometa al principio y seguir hasta que finalmente se ejecuten correctamente las otras funciones.
Se recuerda también que en el estado actual del engine es bastante fácil producir un bloqueo del ordenador si una parte del polígono sale fuera de la pantalla por un extremo que no sea el izquierdo sin que la rutina de corte haya sido mejorada.

LA FUNCION DE IMPRESION DE UN POLIGONO

Sin duda es la más compleja de las que consta el engine 3D; para facilitar su comprensión a continuación se explicarán las tareas que realiza paso a paso:
void PolygonDrawFilled(polygon_type &poligono)

```
{
 //Variables de tipo entero que se usan dentro de la función
 sint n,n2,n3,n4,n5,n6,x,y;
 //Punteros a vertices, se usan para acceder más rapidamente a los datos
 vertex_type *ver1,*ver2,*veraux;
```

La variable *"miny2d"* contendrá la coordenada "Y" donde empieza el polígono, mientras que *"maxy2d"* la coordenada "Y" donde termina. Se deben de inicializar a valores extremos, aunque luego se modifique su valor:

```
 //Coordenadas mínima y máxima en altura del poligono
 float miny2d=10000,maxy2d=-10000;
```

Las variables *"cont"*, *"zbuffcont"*, *"contmx"* y *"contmy"* contendrán, respectivamente, el valor en X, la profundidad y las coordenadas de mapping del punto que se esté imprimiendo en cada momento. *"inc"*, *"zbuffinc"*, *"incmx"* e *"incmy"* contendrán el valor que se incrementa a las variables anteriores al pasar de un punto al siguiente:

```
 //Contador y acumulador
 float inc,cont,zbuffinc,zbuffcont;
 //Contador y acumulador para el texture mapping
 float contmx,contmy,incmx,incmy;
```

La técnica usada para imprimir un polígono relleno, tenga o no texturas, consiste en descomponerlo en una serie de líneas horizontales, todas con un principio y un fin en una coordenada. A continuación se inicializa esa tabla con valores extremos, para luego poder modificarla:

```
 //Se inicializa la tabla de líneas horizontales
 for(n=0;n<maxalto;n++)
 {
  minx2d[n]=10000;
  maxx2d[n]=-10000;
 }
 //Se Añade al final un vértice más al polígono que será el mismo que el
 //primero para que al hacer el bucle quede completamente cerrado
 poligono.vertex[poligono.n_vertex].sx=poligono.vertex[0].sx;
 poligono.vertex[poligono.n_vertex].sy=poligono.vertex[0].sy;
 //Coordena de Zbuffer
 poligono.vertex[poligono.n_vertex].zbuff=poligono.vertex[0].zbuff;
 //Coordena de Texture mapping
 poligono.vertex[poligono.n_vertex].mapx2=poligono.vertex[0].mapx2;
 poligono.vertex[poligono.n_vertex].mapy2=poligono.vertex[0].mapy2;
 //Se Carga ver1 y ver2 con los dos primeros vertices del polígono
 ver1=&(poligono.vertex[0]);
 ver2=&(poligono.vertex[1]);
```

Ahora viene una de las partes más importantes de la función. Consiste en realizar un bucle recorriendo todo el polígono, descomponiéndolo en una serie de líneas horizontales. Las tablas *minx2d* y *maxx2d* contienen dónde empieza y termina cada una de las mencionadas líneas horizontales, mientras que las tablas *minx2dz, maxx2dz, minx2dmx, maxx2dmx, minx2dmy* y *maxx2dmy* contendrán la profundidad en el Z-buffer y las coordenadas de mapping en X e Y del punto inicial y final de cada línea:

```
 //Este bucle convierte el polígono en una serie de líneas horizontales
 for(n=0;n<((poligono.n_vertex));n++)
 {
 //Si el primer vertice está por debajo del segundo los invertimos, para
 //así evitar errores en los cálculos
 if((ver1->sy)>(ver2->sy))
 {
  veraux=ver1;
  ver1=ver2;
  ver2=veraux;
 }
```

Aquí se averigua la primera coordenada "Y" del polígono y la última. Este dato será importante luego, ya que las tablas que contienen el polígono descompuesto en líneas usan estos valores. *"miny2d"* contendrá el valor de índice en las tablas de la primera línea válida del polígono y *"maxy2d"* el de la última línea válida del mismo. Por ejemplo, un polígono cuyo primer píxel esté a una altura de 50 en la coordenada "Y", y el inferior en la 120 será descompuesto en 70 líneas horizontales que estarán contenidas en las tablas desde la posición 50 hasta la 120:

```
 //Limita la altura del polígono
 if((ver1->sy)<(miny2d))
  miny2d=ver1->sy;
 if((ver2->sy)>(maxy2d))
  maxy2d=ver2->sy;
 //Cálculo de los puntos intermedios para componer una línea
 cont=ver1->sx;
 inc=((ver2->sx)-(ver1->sx))/((ver2->sy)-(ver1->sy)+1);
 //Cálculo de puntos intermedios de profundidad para calcular el Z buffer
 zbuffcont=ver1->zbuff;
 zbuffinc=((ver2->zbuff)-(ver1->zbuff))/((ver2->sy)-(ver1->sy)+1);
 //Cálculo de puntos intermedios para texture mapping
 contmx=(ver1->mapx2);
 incmx=(((ver2->mapx2)-(ver1->mapx2))/((ver2->sy)-(ver1->sy)+1));
 contmy=(ver1->mapy2);
 incmy=(((ver2->mapy2)-(ver1->mapy2))/((ver2->sy)-(ver1->sy)+1));
 //Almacena el polígono en la tabla de líneas horizontales
 n5=ver1->sy;
 n6=ver2->sy;
 for(n2=n5;n2<=n6;n2++)
 {
  if(cont<minx2d[n2])
```




```
 {
  minx2d[n2]=cont;
  minx2dz[n2]=zbuffcont;
  minx2dmx[n2]=contmx;
  minx2dmy[n2]=contmy;
 }
 if(cont>maxx2d[n2])
 {
  maxx2d[n2]=cont;
  maxx2dz[n2]=zbuffcont;
  maxx2dmx[n2]=contmx;
  maxx2dmy[n2]=contmy;
 }
 cont+=inc;
 zbuffcont+=zbuffinc;
 contmx+=incmx;
 contmy+=incmy;
 }
 //Se pasa a los siguientes vertices
 ver1++;
 ver2++;
}
```

Aquí viene la segunda parte importante de la función. Una vez que el polígono fue descompuesto en las líneas horizontales se procede a imprimir todas esas líneas. Los datos se extraen de los valores introducidos anteriormente en las tablas. Como se dijo antes, se deben de usar los valores contenidos desde la posición "*miny2d*" hasta la "*maxy2d*" de las tablas:

```
//Pinta en la pantalla el polígono descompuesto en líneas horizontales
//Comprobando el Zbuffer;
if(miny2d<maxy2d)
{
 for(y=miny2d;y<=maxy2d;y++)
 {
```

En esta otra parte del código se extraen los valores de profundidad en el Z-buffer del primer y del último punto de la línea horizontal que se está procesando en cada momento, y se obtiene un valor que, sumado sucesivas veces a la profundidad del primer punto, dará la profundidad del último punto al llegar al final de la línea. Es decir, se usa una técnica de sumas sucesivas para ir de un valor inicial a otro final pasando por todos los valores intermedios:

```
  //Profundidad en Z de cada uno de las líneas horizontales
  zbuffcont=minx2dz[y];
  zbuffinc=((maxx2dz[y]-minx2dz[y])/(maxx2d[y]-minx2d[y]+1));
```

Lo mismo que se ha hecho en el Z-buffer se hace con la coordenada "X" y la coordenada "Y" del mapping:

```
  //Texture mapping de cada una de las líneas horizontales
  contmx=minx2dmx[y];
  incmx=((maxx2dmx[y]-minx2dmx[y])/(maxx2d[y]-minx2d[y]+1));
  contmy=minx2dmy[y];
  incmy=((maxx2dmy[y]-minx2dmy[y])/(maxx2d[y]-minx2d[y]+1));
```

Finalmente, se recorre toda la línea pintándola punto a punto. Un punto sólo será dibujado si está a una profundidad de Z-buffer inferior a la del punto que ocupaba anteriormente su posición:

```
  for(x=minx2d[y];x<=maxx2d[y];x++)
  {
   //Si la profundidad del nuevo punto es menor que la del que ocupaba
   //dicha posición en el fondo de la pantalla se pinta el nuevo punto
   //y se actualiza la profundidad.
   //En caso contrario, no se pintará el punto y tampoco se actualizará
   //la profundidad.
   if(zbuffcont<(*(zbuffer.Lines[y]+x)))
   {
    //PutPixel(x,y,poligono.color); Sin mapeado de texturas
    PutPixel(x,y,GetPixel(((long)contmx),((long)contmy)));
    (*(zbuffer.Lines[y]+x))=zbuffcont;
   }
   zbuffcont+=zbuffinc;
   contmx+=incmx;
   contmy+=incmy;
  }
 }
}
}
```

En la función "*Draw3DObject*" ha vuelto a ser incluida la parte del clipping. Aunque sólo el corte 2D por el extremo izquierdo de la pantalla soporta Z-buffer y mapeado de texturas:

```
if(Polygon_Clip_3dz(obj.polygon[aux]))
 {
  //Pasa las coordenadas 3D a coordenadas 2D de pantalla
  PolygonCalcule2D(obj.polygon[aux]);
  //Cortes 2D. En caso que el número de vértices del polígono tras ser
  //recortado sea igual a 0 no se ejecuta la función PolygonDrawFilled
  //NOTA: Solo el primero de los cuatro cortes realiza también
  //El clip en el mapping y en el Z-buffer
  if(Polygon_Clip_2dx1_Updated(obj.polygon[aux]))
   if(Polygon_Clip_2dx2(obj.polygon[aux]))
    if(Polygon_Clip_2dy1(obj.polygon[aux]))
     if(Polygon_Clip_2dy2(obj.polygon[aux]))
      PolygonDrawFilled(obj.polygon[aux]);
 }
```

El programa principal no presenta grandes modificaciones con respecto al del mes pasado, simplemente pinta el cubo en otras coordenadas de pantalla:

```
for(auxf=1400;auxf>1000;auxf-=2.0)
{
 Draw(ob1,-1800+auxf,150,auxf-400,auxf/40,auxf/55,auxf/60,0);
}
```

Próximo capítulo

Dominada ya la técnica del mapeado de texturas y el recorte de polígonos , lo más difícil ha pasado; ahora queda mejorar el engine con efectos realistas. La iluminación será el tema que se tratará en el próximo número.
Se realizará un ejemplo práctico iluminando nuestro cubo con una fuente de luz. El cubo tendrá texturas en sus caras, por lo que se sumará el efecto de mapeado de texturas al de iluminación.
Otra mejora que también se realizará será la de permitir que una textura de pequeño tamaño se repita un número determinado de veces en una cara de un polígono.



Autor: Santiago Romero

Trabajando en SVGA [II]

Como vimos en la anterior entrega, existen dos posibilidades a la hora de trabajar con gráficos en PC. La primera de ellas consiste en utilizar un modo de vídeo específico bajo unas condiciones predeterminadas al desarrollar nuestra aplicación o juego. Esto es lo que ocurre cuando se decide utilizar el modo 13h, o modos SVGA de 8 bits por píxel. La segunda consiste en adaptarse a las características del sistema durante el proceso de carga de la aplicación.

APLICACIONES ESPECIFICAS

Supongamos que elegimos un modo de 8bpp (p. ej. 640x480x256) y pantalla completa para la realización de nuestra aplicación/juego. En estas condiciones se tiene prediseñado todo el código y gráficos de antemano, sabiendo que nada va a cambiar durante la ejecución de la aplicación (ejemplo: juegos MSDOS 256 colores y aplicaciones DX full screen en 8bpp).

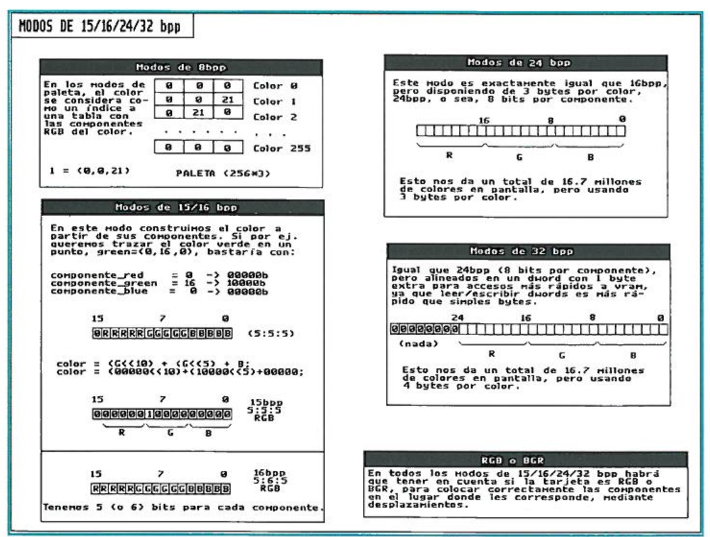
Mediante este método de trabajo limitamos la posibilidad de utilizar aplicaciones *windowed*. Bajo Windows95, si diseñamos una aplicación GDI (que use el *Interface de Dispositivo Gráfico* estándar de Windows), el mismo GDI es capaz de hacer conversión automática al realizar el volcado del bitmap si la profundidad de color (bpp) del DC (contexto de dispositivo) destino es diferente de la origen (mediante BitBlt()). Por supuesto, esto resultará más lento que realizar una sola conversión durante el inicio de la aplicación, como se comentó en la anterior entrega.

ADAPTACION AL DISPLAY

Una segunda posibilidad representa el desarrollar un programa independiente del dispositivo o susceptible de cambios en el display durante la ejecución del mismo.

En diferentes tarjetas los modos *HiColor* (15/16bpp) pueden definirse mediante 5 bits para la componente rojo (R), 5 para la verde (G) y 5 para la azul (B) (5:5:5 = 15 bits), aunque también es muy común encontrar tarjetas que trabajen con un sistema de 5:6:5 (6 bits para la componente verde). Además, la tarjeta puede trabajar con RGB (atendiendo a cómo

FIGURA 1. DIFERENTES FORMATOS DE PIXEL.



Vimos en la anterior entrega los diferentes modos SVGA, así como sus fundamentos a nivel de la propia tarjeta. Este mes desgranamos su tratamiento como mapa de bits, trabajando con las distintas profundidades de color, y realizando conversiones de imágenes entre ellas.

están situadas las componentes dentro del word o dword) o BGR, lo cual obliga a definir diferentes modos de trabajo según las características de la misma, así como saber convertir los gráficos entre estos formatos. Para comprender los diferentes modos vamos a ver cómo trabaja la primitiva PutPixel() dentro de ellos.

MODOS DE 8BPP

Los modos de 8bpp han sido ya comentados durante el curso. Cada elemento del buffer de trabajo (cada byte) constituye un índice (apuntador) dentro de una tabla de componentes RGB (la paleta) que el dispositivo gráfico utiliza para representar cada byte del mapa de bits. Estos modos son rápidos y sencillos, requieren poca memoria (al ser cada elemento de tipo byte) y han sido los más utilizados hasta ahora. Su organización interna puede observarse en las figuras 1 y 2.

```
// tamaño de la pantalla/buffer
#define ANCHO 640
#define ALTO  480

byte buffer[ANCHO*ALTO] ;
void PutPixel_8bpp( x, y, byte color )
{
  buffer[(y*ANCHO)+x] = color ;
}
```

La lectura es igual de sencilla; para el acceso a un elemento (x,y) basta con aplicar una conversión directa de ambas coordenadas sabiendo la anchura lógica de cada *scanline* del buffer/pantalla.

MODOS DE 16BPP 5:5:5

El modo 5:5:5 no es exactamente de 16 bits por píxel, sino de 15 (5 bits para RED, 5 para GREEN y 5 para BLUE suman un total de 15 píxeles). El bit más significativo puede ignorarse a la hora de trazar o leer píxeles (*modo de 15bpp* o *32kColour*). En las figuras 1 y 2 podemos ver un esquema interno de este modo. Atendiendo al mismo:

```
// Colocación de cada componente:
#define MAKERGB_15bpp(r,g,b) (word) ((r<<10)|(g<<5)|(b))

word buffer[ANCHO*ALTO] ;
```



```
void PutPixel_15bpp( x, y, int r, int g, int b )
{
  buffer[((y*ANCHO)+x)] = MAKERGB_15bpp(r, g, b) ;
}
```

Lo primero que notamos en el listado anterior es que el buffer se ha declarado usando componentes de tipo *short* (2 bytes), ya que en 15bpp cada píxel es representado por 2 bytes (16/8=2). Siguiendo con el ejemplo, puede observarse que se ha definido una macro *MAKERGB* que acepta las 3 componentes (con valores entre 0 y 31 cada una (5 bits)), y construye un *word* (color/píxel) mediante desplazamientos y operaciones lógicas OR (|). Vamos a examinar detenidamente dicha macro:
Supongamos que queremos trazar un píxel de valores RGB (16,9,1). Con esos datos, ejecutamos una llamada a la función PutPixel:

```
PutPixel_15bpp( x, y, 16, 9, 1 );
```

Para la llamada a la macro MAKERGB se tiene que:

```
r = 16 = 10000b
g =  9 = 01001b
b =  1 = 00001b
```

El contenido final del byte a escribir (5:5:5) ha de quedar de la siguiente manera (debido a la organización interna del modo):

```
(555RGB): bit 15 -> 0rrrrrgggggbbbbb <- bit 0
```

Aprovechando los desplazamientos (<<) de bits y la operación OR (que permite introducir valores sin modificar los bits existentes aparte de los activados ya en la variable), se deduce que:

```
color = (r<<10) | (g<<5) | (b) ;
```

Los desplazamientos dejan cada componente en el lugar que le corresponde dentro del *word* final, y las operaciones OR realizan la composición de todas las componentes en un solo elemento de memoria.

Cada elemento del buffer de trabajo (cada byte) constituye un índice (apuntador) dentro de una tabla de componentes RGB

El resultado final es, pues, 0100000100100001b, que es el parámetro pasado a la función PutPixel. Esta función se encarga simplemente de almacenar este valor en el elemento correspondiente del array de tipo *word*. Nótese que las funciones que se están analizando son todos en formato RGB. Para trazar el mismo píxel en una tarjeta BGR basta con cambiar los valores de los *shifts* a *((b<<16)|(g<<8)|(b))*, aunque luego veremos funciones de conversión para este tipo de modos.

MODOS DE 16BPP 5:6:5

El modo 5:6:5 constituye el modo *HiColour/64KColour*, y es exactamente igual al anterior (figura 1), con la única excepción de que ahora se utilizan 5 bits para RED, 6 para GREEN (rango 0-63) y 5 para BLUE. Basta pues con cambiar los valores de los desplazamientos:

```
// Colocación de cada componente:
#define MAKERGB_16bpp(r,g,b) (word) ((r<<11)|(g<<5)|(b))
```

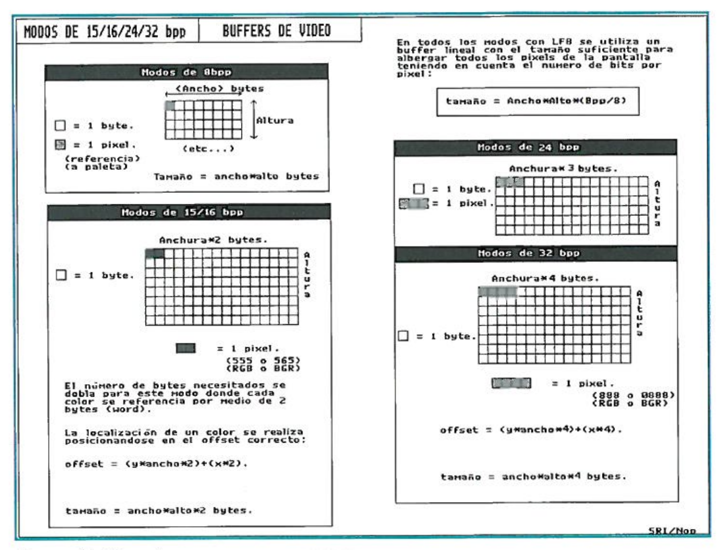


FIGURA 2. TAMAÑOS DEL BUFFER DE VIDEO.

La rutina PutPixel() realiza el mismo trabajo que la anterior, almacenar el *word* resultante en el buffer de trabajo/vram.

MODOS DE 24BPP 8:8:8

En las figuras 1 y 2 puede observarse la organización del modo de 24bpp, en el que se aprecia la utilización de 8 bits por componente, lo que requiere (24bpp/8=3) 3 bytes para describir un color. Al no existir ningún tipo de datos de 3 bytes para almacenar un color puede optarse por usarse un *long o int* (4 bytes) o 3 bytes individuales.

a) Tratado individual de las componentes:

```
#define NUMBPP 3

char buffer[ANCHO*ALTO*NUMBPP] ;

void PutPixel_24bpp_1( x, y, byte r, byte g, byte b )
{
  long offset = ((y*ANCHO)+x)*NUMBPP;
  buffer[offset]   = r;
  buffer[offset+1] = g;
  buffer[offset+2] = b;
}
```

Se define un array con el tamaño necesario para todos los elementos (ancho*alto*3bytesporpixel), se calcula el offset perteneciente a dicho píxel ((y*ancho+x)*3), y se almacenan en memoria las 3 componentes del color especificado.

b) Aglutinación de las componentes:

```
char buffer[ANCHO*ALTO*NUMBPP] ;

// Colocación de cada componente:
#define MAKERGB(r,g,b) (dword) ((r<<16)|(g<<8)|(b))

void PutPixel_24bpp_2( x, y, dword color )
{
  long offset = ((y*ANCHO)+x)*NUMBPP;
  buffer[offset]   = (byte) (r>>16) & 255;
  buffer[offset+1] = (byte) (g>>8)  & 255;
```




```
  buffer[offset+2] = (byte) (b & 255);
}
```

En esta rutina pasamos el color ya compuesto como parámetro para descomponerlo en los bytes individuales. Para ello desplazamos la componente deseada a los 8 últimos bits del *word* y hacemos un AND con 255, para poner a cero todos los bits del long excepto los 8 últimos (255=11111111b), que convertimos a byte para escribir en el buffer (aunque el *typecast* ya estaría implícito en la escritura).

VENTAJAS Y DESVENTAJAS

Los modos de 256 colores tienen grandes desventajas, como una mayor dificultad de creación de sombreados (efectos de luces/sombras), para los cuales será necesario definir una paleta preparada para diferentes tonalidades (por ejemplo, 64x4 tonalidades), o utilizar elaboradas *look-up-tables* (*LUT*, o tablas precalculadas), donde podamos encontrar equivalentes brillantes/oscuros de los 256 colores. Esto lo veremos en la próxima entrega, junto con rutinas de *"best match"*.
También nos encontramos con el típico problema de paletización de bitmaps. Imaginemos 2 sprites cada uno de ellos con su paleta propia y se desea visualizarlos juntos en pantalla. Al no tener idéntica paleta deberemos proceder a construir una paleta óptima para ambos y *mapear* los colores de los sprites a esta paleta (esto es engorroso, por ejemplo, para un juego aunque no lo hagamos en tiempo real).

Escribir un dword en memoria puede no ser tan rápido si no lo hacemos adecuadamente

Precisamente lo contrario ofrecen los modos de 16bpp o más. A cambio de requerir una mayor cantidad de videomemoria (y, por lo tanto, menor número de fps al haber más datos, más pérdidas de caché, etc), ofrecen la individualidad de cada píxel en pantalla, lo que nos permitirá modificar la intensidad/color de un píxel sin modificar los colores similares en la imagen (es decir, ya no es posible modificar la paleta puesto que no existe). Esto da la posibilidad de realizar efectos de luces/sombras añadiendo/restando valores constantes a todos los píxeles de un sprite en pantalla, como los conocidos *flares* que pueden verse en muchas demos/juegos.

OPTIMIZACIONES SIMPLES

La mejor optimización que puede realizarse para acelerar el trazado de gráficos es, obviamente, no utilizar llamadas a la rutina PutPixel para la realización de gráficos complejos (sino funciones específicas). En la realización de estos, siempre que sea posible, se deberá de *tratar de aglutinar colores en dwords:*

- Si sabemos que todos los bitmaps/sprites de nuestra aplicación son de una anchura múltiplo de 4, al trazarlos se leerán 4 bytes (un *dword*) y se escribirá éste en el buffer destino.
- En el caso de 15/16bpp, tratar de escribir 2 píxeles cada vez utilizando para ello un *dword* (2 píxeles x 2 bytes = 4 bytes).
- En los modos de 24bpp, escribir (si es posible) 4 píxeles cada vez, utilizando para ello 3 *dwords* (4 píxeles x 3 bytes = 12 bytes=3dw).

De esta manera, además de hacer accesos a memoria más rápidos, se estará desenrrollando el bucle a 1/4, 1/2 o 1/3 de las iteraciones.

Modos de 32bpp 0:8:8:8

Este modo es similar al anterior: posee 8 bits para cada componente, pero cada color se representa por 4 bytes (un dword), ignorando el byte superior de los 4 (ORGB). Esto se hizo así para aprovechar que el bus de los PC's escribe de manera más rápida y eficiente un DWORD que un BYTE (aunque pueda parecer que debería tardar más tiempo), debido a su arquitectura interna.

```
dword buffer[ANCHO*ALTO] ;

// Colocación de cada componente:
#define MAKERGB(r,g,b) (dword) ((r<<16)|(g<<8)|(b))

void PutPixel_32bpp( x, y, dword color )
{
  buffer[((y*ANCHO)+x)] = color;
}
```

Es el modo más sencillo de trabajar (aparte del de 8bpp), el más completo (16,7 millones de colores) y el que más memoria necesita (hasta 4 veces más).

OPTIMIZACIONES AVANZADAS

a) Alineamiento de datos en memoria.
Escribir un *dword* en memoria puede no ser tan rápido si no lo hacemos adecuadamente. Si la escritura/lectura no se realiza en una posición de memoria múltiplo de 4 (0, 4, 8...) es decir, si no está alineado con el tamaño del *dword* (por ejemplo, escribir en Offset=1/2/3), sufriremos una penalización de 3 ciclos de reloj por acceso. Al trazar un sprite por medio de instrucciones MOVSD (o por medio de un bucle for() con aritmética de punteros a long); si la posición del sprite es, por ej., (1,0) (desalineado), se nos penalizará con 3 ciclos de reloj *por cada elemento que escribamos a memoria* (por eso existe en los compiladores una opción de alineación de datos a 1/2/4 bytes).
Una forma de solucionar esto es tener 4 rutinas diferentes ((DrawSpr0 a DrawSpr3) que dependiendo del resto de la coordenada X destino del sprite (0,1,2,3)), escriban los píxeles desalineados como bytes y el resto como *dwords*. Se debe llamar a una u otra en función de la coordenada x del sprite (esto, es, *NumRutina = SpriteX & 3*, o *NumRutina = SpriteX % 4 ;* (el resto de dividir por 4)).
La primera de ellas, DrawSpr0, debe ser llamada cuando la coordenada. X es múltiplo de 4 (spriteX & 3 da 0 cuando ocurre esto). Ésta se encarga de volcar *dwords* sin preocuparse, pues el destino está alineado a 4.
A modo de ejemplo, supongamos que trazamos el sprite en (3,0). La rutina llamada será DrawSpr3, que deberá escribir 1 byte del sprite en memoria y, en ese momento, el siguiente píxel ya estará en SpriteX = 4 (alineado), por lo que podemos volcar el resto del sprite con *dwords* (cuidado con los últimos bytes del mismo). De similar manera habremos de crear DrawSpr1 (escribir primero 3 bytes para alinear, y luego *dwords*) y DrawSpr2 (escribir 2 bytes y luego *dwords*).

b) Uso de FPU o MMX.
Puede usarse tanto la FPU (coprocesador matemático) como las instrucciones MMX para copiar 64 bytes de golpe (más rápido que el equivalente en *movsd*) de una posición de memoria a otra. Veamos un ejemplo de copia con FPU:

```
fild qword ptr [eax]
fild qword ptr [eax+8]
fild qword ptr [eax+16]
fild qword ptr [eax+24]
fild qword ptr [eax+32]
fild qword ptr [eax+40]
fild qword ptr [eax+48]
fild qword ptr [eax+56]
fxch st(1)
fistp qword ptr [ebx+48]
fistp qword ptr [ebx+56]
```



fistp qword ptr [ebx+40]
fistp qword ptr [ebx+32]
fistp qword ptr [ebx+24]
fistp qword ptr [ebx+16]
fistp qword ptr [ebx+8]

El anterior fragmento de código copia 64 bytes desde la dirección apuntada por EAX a EBX. Las instrucciones *FILD* (*Fpu Integer LoaD*) almacenan 8 bytes cada una en la pila del coprocesador, y las instrucciones *FISTP* (*Fpu Integer Store and Pop*) saca el último valor de la pila de la fpu y lo almacena (como un *integer*) en la posición indicada. La primera instrucción de almacenamiento (*fistp*) se ha intercambiado por la segunda (+56 por +48), para no producir 2 accesos seguidos a la misma posición de la pila y, por tanto, no obtener ninguna penalización. Este tipo de copia es muy recomendable con buffers de volcado grandes (640x480x16bpp); tan solo hay que utilizar un bucle en incrementemos EBX y EAX en 64 hasta realizar el volcado.

LISTADO 1. FUNCIONES DE CONVERSION 32BPP->XBPP

```
//-----------------------------------------------
short Convert32To15( long color )
{
   long result, r, g, b;

   r = (color>>16) & 255;
   g = (color>>8)  & 255;
   b = color & 255;

   r = r>>3;
   g = g>>3;
   b = b>>3;

   result = (r<<10) | (g<<5) | b;
   return( (short) result );
}

//-----------------------------------------------
short Convert32To16( long color )
{
   long result, r, g, b;

   r = (color>>16) & 255;
   g = (color>>8)  & 255;
   b = color & 255;

   r = r>>3;
   g = g>>2;
   b = b>>3;

   result = (r<<11) | (g<<5) | b;
   return( (short) result );
}

//-----------------------------------------------
long Convert32To24( long color )
{
  return( color & 0x00FFFFFF ) ;
}
```

CONVERSION ENTRE LOS DIFERENTES FORMATOS

Para convertir todos los formatos de profundidad de color entre sí tenemos 30 posibilidades distintas (8->15, 8->16, etc.), así que una forma muy práctica de hacer las conversiones es convertir el color desde cualquier formato a 32bpp y después de 32bpp al formato deseado. De esta manera, sólo necesitamos 7 funciones de conversión, 4 de ellas desde Xbpp a 32bpp, y las otras 3 desde 32bpp a Xbpp. No vamos a convertir de 32bpp a 8bpp porque eso requiere un algoritmo de cuantización de color (tenemos que crear una paleta para el bitmap, y sólo dispondremos de 256 colores para ello).

CONVERSION 32BPP->XBPP

Puesto que son las más sencillas, vamos a ver primero las conversiones desde 32bpp a cualquier otro formato. Como puede verse en el listado 1, se basa simplemente en desplazar cada componente perdiendo en el >> los bits que le sobran. Es decir, si queremos pasar un RED de 8 bits a 5 bits, bastará con desplazarla >>(8-5) = >>3.

CONVERSION XBPP->32BPP

En el listado 2 pueden observarse los pasos necesarios para cada una de estas conversiones. Para pasar de 8bpp a 32bpp no hay ninguna dificultad, tan sólo hay que pasarle las componentes (de 8 bits, no de 6) y colocarlas en su lugar adecuado en el *dword*. Se ha optado por pasarlas compuestas dentro de un *long* (similar a 24bpp).
Un sencillo problema surge en las conversiones de 5 o 6 bits a 8. Como puede verse en *Convert15bppTo32bpp()*, no basta con desplazar la componente 3 bits (8-5=3), porque haciendo esto dejamos 3 bits a cero en la parte inferior del byte :

11111 red << 3 = 11111000

Esos 3 ceros son incorrectos y los hemos de cubrir con la parte superior de la componente (antes de desplazarla), de esta manera (5-3=2):

Red = (red<<3) | (red>>2) ;

Y de la misma manera con el resto de componentes y conversiones.

LISTADO 2. FUNCIONES DE CONVERSION XBPP->32BPP

```
//-----------------------------------------------
long Convert8To32bppRGB( long colorRGB )
{
   long result, r, g, b;

   r = (colorRGB>>16) & 255;
   g = (colorRGB>>8)  & 255;
   b = colorRGB & 255;
   result = (r<<16) | (g<<8) | b;

   return( result );
}

//-----------------------------------------------
long Convert15To32bppRGB( short color )
{
   long result, r, g, b;

   r = (color>>10) & 31;
   r = (r<<3) | (r>>2);
   g = (color>>5) & 31;
   g = (g<<3) | (g>>2);
   b = color & 31;
   b = (b<<3) | (b>>2);

   result = (r<<16) | (g<<8) | b;
   return( result );
}

//-----------------------------------------------
long Convert16To32bpp( short color )
{
   long result, r, g, b;

   r = (color>>11) & 31;
   r = (r<<3) | (r>>2);
   g = (color>>6) & 63;
   g = (g<<2) | (g>>4);
   b = color & 31;
   b = (b<<3) | (b>>2);

   result = (r<<16) | (g<<8) | b;
   return( result );
}

//-----------------------------------------------
long Convert24To32bpp( long color )
{
   return( color & 0x00FFFFFF ) ;
}
```

CONVERSION RGB<->BGR

Como ya se ha comentado, la conversión entre ambos tipos de formato equivale a cambiar el orden de los desplazamientos dentro del *dword* final. El listado 3 refleja las dos funciones de conversión.



LISTADO 3. CONVERSION RGB<->BGR

```
//-----------------------------------------
long Convert32bppRGBToBGR( long color )
{
   long result, r, g, b;

   r = (color>>16) & 255;
   g = (color>>8)  & 255;
   b = color & 255;
   result = (b<<16) | (g<<8) | r;
}

//-----------------------------------------
long Convert32bppRGBToBGR( long color )
{
   long result, r, g, b;

   r = color & 255;
   g = (color>>8)  & 255;
   b = (color>>16) & 255;
   result = (r<<16) | (g<<8) | b;

}
```

AVERIGUAR EL ESTADO DEL DISPLAY

Para averiguar el formato del buffer de vídeo, y convertir nuestras imágenes al mismo, podemos solicitar al sistema que nos indique las máscaras (patrones de bits) de las distintas componentes de color, además del estado del display (8/16/24/32bpp).
Para averiguar el número de bits por píxel en que se está trabajando bajo Windows (bajo DOS lo seleccionamos nosotros mismos) simplemente hemos de obtener la información del contexto de dispositivo mediante:

numbpp = GetDeviceCaps(hdc, BITSPIXEL);

Respecto al tipo de modo (555, 565, RGB, etc.), para Vesa/MSDOS, en el *ModeInfoBlock* (de cada modo de vídeo), disponemos de las siguientes variables para realizar la identificación del modo de display:

char RedMaskSize, RedFieldPosition;
(e igual con Green y Blue)

La primera indica el número de bits para esa componente (por ejemplo, *RedMaskSize* es 5 para el modo 5:5:5), y la segunda el lugar donde está situada esa componente (para el shift). Para *RedFieldPosition* en 5:6:5 obtenemos 11 para RGB y 0 para BGR. De esta manera, podemos crear fácilmente una sóla rutina de conversión con estos datos.
En el caso de DirectX, como resultado de solicitar una descripción de la *DDSURFACE*, obtendremos 4 máscaras de 32 bits (RedMask, GreenMask, etc), que nos permitirán identificar el modo de display activo.
Por ejemplo, para 565RGB se devuelve en la redmask el valor 0xf800. Este valor pasado a binario es 1111100000000000b que, como puede verse, corresponde a la posición de la componente RED en el color. Greenmask (también devuelta) tomaría el valor 0000011111100000b. De esta manera, podemos identificar el tipo de buffer de vídeo a partir de las posiciones (*xxSize*) y de los valores (*xxPosition*).
Con toda esta información, para convertir la imagen, por ejemplo, de 15bpp a 24bpp, podemos utilizar un bucle como el siguiente :

```
for( y=0; y<altura; y++ )
 for( x=0; x<anchura; x++ )
 {
  color = GetPixel(x, y, origen);
  color = 32To24bpp( 15To32bpp( color ) );
  PutPixel(x, y, color, destino) ;
 }
```

CONVERSION ELEGANTE

Un método de conversión más elegante que el de disponer de tan elevado número de rutinas es aprovechar directamente los datos de las máscaras devueltos por el sistema y contar uno mismo (con un bucle de conteo de bits) las posiciones de los mismos, realizando una sóla rutina de conversión de imágenes a la que se le pasen como parámetros las máscaras origen, destino y el color a convertir, pero el método expuesto en este artículo ha de servir para lo que estaba pensado: mostrar de manera sencilla las conversiones. Para ello escaneamos desde el final (bit 0) hasta encontrar un bit distinto de 0, haciendo lo mismo desde el principio (último bit). La diferencia entre estos 2 valores más 1 (bit más alto - bit menos alto encontrados != 0 + 1) constituye el número de bits para esa componente.

Podemos identificar el tipo de buffer de vídeo a partir de las posiciones xxSize y de los valores xxPosition

Así, para un *redmask* de 0x7c00 (0111110000000000), encontramos que el bit menor a 1 es el número 10, y el más alto es el 14. Así, el número de bits para esta componente (xxSize) es 14-10+1=5, y la posición base (xxBase) para el desplazamiento << es el la del bit menor (10).
Con este sístema podemos, de una manera general, controlar cualquier tipo de organización de videomemoria, haciendo conversiones con estos datos. Al obtenerse RedSize y RedBase en DirectX nos encontramos en el mismo caso que en Vesa 2.0 (donde teníamos *RedMaskSize* y *RedFieldPosition*, con exactamente el mismo significado), de manera que podemos realizar el mismo tipo de conversión abstracta.

LOCK(), UNLOCK() Y DIRECTX

Las rutinas de dibujo que proporciona DirectX para el trazado (blt()) de gráficos resultan muy rápidas si existe aceleración hardware en el sistema. Si no es así, resulta muy sencillo escribir nuestras propias rutinas de volcado de bmps/sprs, tal y como hemos explicado hasta ahora. Para ello, al dibujar en una Surface (similar a un buffer lineal), tan solo tenemos que bloquearla (*lock()*) para escribir en ella como en cualquier otro buffer. Tras el trazado, la desbloqueamos (*UnLock()*) para devolver su control a DX.

Para averiguar el número de bits por píxel bajo Windows hemos de obtener la información del contexto de dispositivo

Al escribir en la dirección devuelta por el Lock() (**lpSurface*), no podemos utilizar el ancho lógico de pantalla, sino que hay que usar el valor especificado por DirectX como ancho de la Surface (ya que no es obligatoriamente el ancho que le pedimos):

```
// usar previamente Surface->lock()
PutPixelDX( x, y, color )
{
 char *Vram = (char *) surface->lpSurface ;
 dword pitch = surface->lPitch ;

 Vram[ (y*pitch)+x ] = color ;
}
```

Próxima entrega

Todo lo visto en esta entrega nos permite cerrar la explicación sobre trabajo en SVGA. Aprovechamos para agradecer a Russ Williams (r.g.p) su ayuda prestada en este tipo de conversiones.
En la próxima entrega se comentará la rutina de BestMatch() para cuantización de color, así como la aplicación de filtros a imágenes (Softens, Blurs...). La rutina de BestMatch servirá también para crear efectos de luces con LookUpTables bajo 8bpp.

Con 

hemos demostrado que

www.divgames.com

cualquiera puede hacer juegos

1ª EDICIÓN AGOTADA



¿cualquiera?

MÁS DE **250.000** COPIAS DE LA VERSIÓN DE EVALUACIÓN





- **15** Videojuegos completos.
- **2000** Gráficos inéditos en 2D y 3D.
- **500** Páginas de ayuda electrónica.
- **1000** Efectos de sonido.
- **348** Páginas de manual de usuario.
- **25000** Horas de desarrollo para crear DIV.

(bueno... cualquiera no)





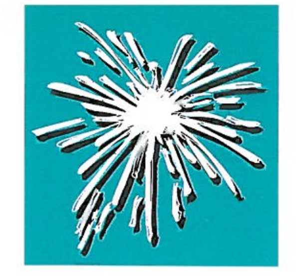
Porque con DIV Games Studio podrás realizar videojuegos comerciales libres de royalties. La imaginación y la creatividad la pones tu, DIV se encarga del resto. Seguro que nunca has visto nada igual.

GANA 1.000.000 Ptas

Con DIV podrás participar en el Concurso de Programación de Videojuegos con DIV, organizado por PC Actual. Y optar a grandes premios y la comercialización de los productos ganadores.



Alfonso Gómez 42, nave 112
28037 Madrid. España
Tel: (91) 3.04.06.22
Fax: (91) 3.04.17.97
e-mail: hammert@studios.com



DEMOSCENE ACTUAL

Nivel: Avanzado

Capítulo 13º

Autor: Pedro Antón

Escribir texto en modo gráfico

Como ya dijimos al comienzo de este curso, abordaremos los problemas de los lectores con especial interés. En esta entrega, una vez más, trataremos una nueva petición de un lector, creando una clase que nos permita escribir texto en pantalla de forma cómoda y sencilla.

El tratamiento que reciben los caracteres es similar al que recibirán los *"sprites"*, aunque ese tema será examinado en un próximo artículo, por ser otra de los peticiones recibidas en esta sección.

DELIMITANDO LA FUENTE

Existen, al menos, dos formas de dibujar una fuente en modo gráfico.
La primera de ellas consiste en identificar y almacenar las figuras geométricas, que conforman la fuente, relativas a un origen. Es decir, supongamos que tomamos como referencia la esquina superior izquierda del carácter; éste será el punto (0,0). Si empleamos números reales para definir la fuente, la esquina inferior derecha será el punto (1,1). Para definir un carácter bastará con almacenar la posición de cada punto característico de éste, relativo a estas dimensiones, junto con la definición de figura geométrica a emplear. En la figura 1 se pueden observar, con mayor claridad, los puntos característicos a los que nos referimos.
Se observan dos figuras geométricas diferentes, tres líneas y una circunferencia. Por tanto, puede definirse dicho carácter como:
Línea desde (0.6,0) hasta (0.8,0)
Línea desde (0.8,0) hasta (0.8,1)
Línea desde (0.8,1) hasta (1,1)
Circunferencia con centro (0.45,0.65) y radio 0.4
A estos datos basta con añadir el ancho y alto del carácter deseado, con lo que la fuente queda perfectamente definida. Para conseguir el tamaño adecuado de la fuente basta con multiplicar los valores de las coordenadas *x* almacenados por el ancho de la fuente deseado, y los de las coordenadas *y*, por el alto. Este tipo de definición es muy útil cuando se quieren emplear los mismos caracteres en distintas resoluciones pero, como se puede comprobar, únicamente nos permite modificar el color de las líneas a trazar. Además, dependiendo de la complejidad de la fuente, dibujar este tipo de texto puede resultar bastante tedioso para el procesador.
La segunda de las formas posibles de escribir texto en modo gráfico es similar a la que emplea la BIOS del PC, para escribir un carácter en los modos de texto disponibles.
En dichos modos, la BIOS tiene almacenada la definición de la fuente de texto a emplear y éstos son adecuadamente interpretados por el generador de caracteres, interpretación que es realizada por una parte del hardware de vídeo.
En nuestro caso, debemos crear una rutina que interprete los datos de la definición de la fuente convenientemente; por ello, no nos limitaremos a usar las fuentes de la BIOS. En la figura 2 se observa la definición de la fuente que vamos a emplear.
Como es de esperar, la fuente empleada es un fichero gráfico. Por lo tanto, es posible dibujar cualquier fuente por llamativa que ésta sea. Al fin y al cabo, ése es nuestro propósito.
La segunda forma consiste, entonces, en copiar el gráfico correspondiente al carácter que se desea escribir, tomando un color como transparente. En la figura, se puede ver cómo el negro es la máscara o color transparente. Es posible sobrecargar la función de escritura y hacer distintos tipos de volcado, pero de momento, esto se deja como ampliación para el lector.

EL CLIPPING

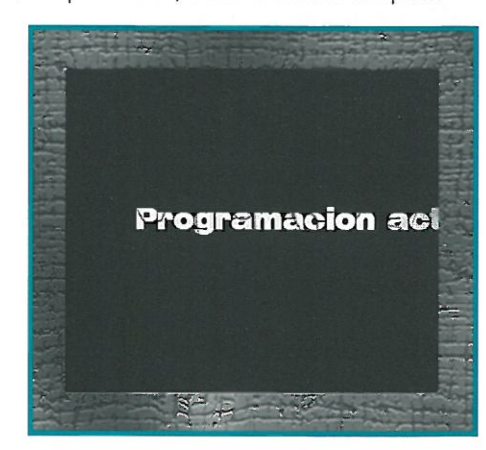
Entremos de fondo en esta última forma de escribir texto en pantalla. Será imprescindible implementar *clipping* o recorte a nuestras fuentes. Si un texto se sale del campo de visión o de la zona de pantalla definida como tal, debemos eliminar o bien la parte no visible del carácter o caracteres que se encuentran fuera del campo de visión, o bien el carácter completo.

LISTADO 1

```
dword Aux   = 0;
int  Wide   = FontW-1; // Ancho de la fuente -1.
int  Height = FontH-1; // Alto de la fuente -1

//Esta fuera?
 if (x>ClipX2)        return;
 if (y>ClipY2)        return;
 if ((x+Wide)<ClipX1)  return;
 if ((y+Height)<ClipY1) return;

//Clipping ...
//... izqdo ...
 if (x<ClipX1)
   {
    Aux    = ClipX1 - x;
    Wide   -= Aux;
    LettOff += Aux;
    x      = ClipX1;
   }
//... superior ...
 if (y<ClipY1)
   {
    Aux    = ClipY1 - y;
    Height -= Aux;
    LettOff += Font.hres*Aux;
    y      = ClipY1;
   }
//... derecho ...
 Aux = x+Wide;
 if (Aux>ClipX2) Wide -= Aux-ClipX2;
//... y abajo
 Aux = y+Height;
 if (Aux>ClipY2) Height -= Aux-ClipY2;
```





LISTADO 2

```
/*
  LeaRNWaRe code by Crom / Spanish Lords 1998

  Objetivo   : Creaccion de una clase para escribir
               textos en modo grafico.
  Autor      : Pedro Ant¢n Alonso. crom@ergos.es
  Plataforma : MS DOS.
  Compilador : Watcom C++ 32 bits - DOS

*/
#ifndef _FONT_CLASS_
#define _FONT_CLASS_
#include "newtypes.h"
#include "pcx.h"

class CFont
{
public:
  int    FontW,FontH;  // Dimensiones de la fuente.
  PCX    Font;
  int    ClipX1;       // El clipping de la fuente.
  int    ClipY1;
  int    ClipX2;
  int    ClipY2;
private:
  dword  FontOff [95]; // Offset de cada letra seg£n al ancho y el alto.
//Funciones de la clase.
public:
  boolean Load    (char *NameFile,int Wide,int Height);
  void   SetClip  (int x1,int y1,int x2,int y2);
  void   write    (dword *Where, char *Text,int x,int y);
  void   End      ();
private:
  void   PutLetter (dword *Where,dword LettOff,int x,int y);
};
#endif
```

Eliminar el carácter completo es la forma más sencilla de recortar el texto; no es nada elegante, pero será suficiente en aquellos casos donde la mayoría del texto escrito se encuentre dentro del campo de visión.

Recortar el carácter que se va a escribir en pantalla y que no se visualiza completamente es la fórmula elegida en el ejemplo que acompaña a este artículo; de esta forma, nos será posible, de una manera extremadamente sencilla, realizar el desplazamiento de un texto, de un lado a otro del campo de visión (*scroll*).

Si analizamos el problema, observamos que tenemos los siguientes datos:

- Dirección de comienzo del carácter que vamos a escribir.
- Ancho y alto del carácter.
- Ancho del formato donde están almacenadas la fuentes.
- Ancho del formato de visualización (modo de vídeo).
- Coordenadas del recorte.

Implementar el *clipping* será sencillo. En primer lugar, se debe comprobar si el carácter a recortar está fuera de la zona de visión. En caso afirmativo, no es necesario aplicar el algoritmo de recorte, no se pintará nada. A continuación comprobamos las zonas de recorte:

Zona izquierda. Si la coordenada **x** del carácter a poner es menor que la coordenada **x** del recorte de la zona izquierda, se calcula la cantidad a recortar, para restarla del ancho y sumarla a la dirección de comienzo de éste; y se actualiza la coordenada **x** del carácter al valor de la coordenada **x** del recorte de la zona izquierda.

Zona superior. Si la coordenada **y** del carácter a poner es menor que la coordenada **y** del recorte de la zona superior, se calcula la cantidad a recortar, para restarla del ancho y actualizar la dirección de comienzo de éste; finalmente, se actualiza la coordenada **y** del carácter al valor de la coordenada **y** del recorte de la zona superior.

Para definir un carácter bastará con almacenar la posición de cada punto característico de éste

Zona derecha. Si el valor máximo de la coordenada **x** del carácter a poner es mayor que la coordenada **x** del recorte de la zona derecha, se actualiza el ancho del carácter, restando la cantidad a recortar.

Zona inferior. Si el valor máximo de la coordenada **y** del carácter a poner es mayor que la coordenada **y** del recorte de la zona inferior, se actualiza el alto del carácter, restando la cantidad a recortar.

El listado 1, muestra la implementación del recorte expuesto.

LA CLASE CFONT

La clase CFont gestiona el manejo de las fuentes para ser utilizadas en nuestros programas. A continuación, mostraremos las funciones y variables miembros de dicha clase.

FIGURA 1. LETRA "D" COURIER NEW

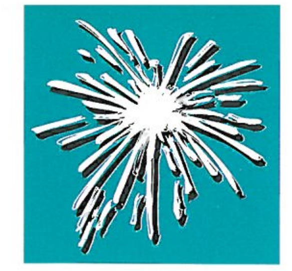
En la definición de la clase nos encontramos, como viene siendo habitual, con dos tipos de variables, públicas y privadas.

El primer tipo almacena la definición básica de la fuente, es decir, el Alto y el Ancho (*FontW, FontH*), así como los datos en sí mismos de la fuente, en formato gráfico (*PCX Font*) y las coordenadas de recorte que definen la zona de visión (*ClipX1, ClipY1, ClipX2, ClipY2*).

En cuanto al segundo tipo, las variables privadas, únicamente nos encontramos con una matriz de 95 datos de 4 bytes cada uno, (*dword FontOff [95]*), donde se almacena el desplazamiento (*offset*) de cada uno de los caracteres que soporta la clase. Esto quiere decir que la clase permite escribir cualquier carácter entre el 32 (el espacio) y el 128. El listado 2 muestra la definición de la clase.

A continuación comentaremos las funciones miembro:

CFont::Load. Esta función debe ser llamada para iniciar el funcionamiento de la clase y se encarga de almacenar los datos que definen la fuente, así como de calcular el offset de cada carácter. Para ello, necesita conocer el ancho y el alto de la fuente definida, por tanto, los parámetros de entrada de esta



LISTADO 3

```
/*
        LeaRNWaRe code by Crom / Spanish Lords 1998

  Objetivo   : Creaccion de una clase para escribir textos en modo
               grafico.
  Autor      : Pedro Antón Alonso. crom@ergos.es
  Plataforma : MS DOS.
  Compilador : Watcom C++ 32 bits - DOS
                                                          */
#ifndef _FONT_CLASS_
#define _FONT_CLASS_
#include "newtypes.h"
#include "pcx.h"

class CFont
{
public:
  int    FontW,FontH;  // Dimensiones de la fuente.
  PCX    Font;
  int    ClipX1;       // El clipping de la fuente.
  int    ClipY1;
  int    ClipX2;
  int    ClipY2;
private:
  dword  FontOff [95]; // Offset de cada letra según al ancho y el
alto.
//Funciones de la clase.
public:
  boolean Load    (char *NameFile,int Wide,int Height);
  void    SetClip (int x1,int y1,int x2,int y2);
  void    write   (dword *Where, char *Text,int x,int y);
  void    End     ();
private:
  void    PutLetter (dword *Where,dword LettOff,int x,int y);
};
#endif
```

función serán el nombre del fichero y el ancho y alto de los caracteres. Llegado este punto, conviene destacar que la definición de la fuente incluye unas líneas verticales que delimitan el tamaño de cada carácter. Éstas deben ser contempladas, tanto en el momento de implementar la clase como a la hora de diseñarla. Destacar también que la implementación de esta función debe calcular, para conocer el desplazamiento de cada carácter, el número de éstos tanto en horizontal como en vertical.

CFont::SetClip. Esta función permitirá modificar las zonas de recorte a nuestro antojo, cambiando las variables de ámbito público que almacenan dicha zona. Esto puede ser especialmente útil para las aplicaciones que empleen distintas zonas de visión.

La clase CFont gestiona el manejo de las fuentes para ser utilizadas en programas

CFont::PutLetter. Es la función que se encarga de recortar la fuente, si es necesario, y de escribirla convenientemente en pantalla. Por tanto, modificándola o sobrecargándola se pueden implementar distintas formas de escribir la fuente, como cursiva, *blending*. La primera parte es la rutina de *clipping*; el resto se encarga de escribir el carácter empleando como mascara el valor 0. Por tanto, si se desea sobrecargar la función, implementando nuevas formas de escritura, puede resultar conveniente escribir la rutina de recorte como una función miembro de clase evitando así reescribirla en cada implementación.

El tratamiento que reciben los caracteres es similar al que recibirán los "sprites"

Veamos cómo trabaja el cuerpo de la función: una vez recortado del carácter, convenientemente, tenemos la coordenada (**x, y**) donde vamos a colocarlo, el ancho y el alto del carácter que vamos a escribir y la ubicación del carácter dentro de la definición de los datos de la fuente (offset del fichero PCX). Con estos datos, direccionamos un puntero a la dirección de comienzo de la definición del carácter (*pFonts*) e inicializamos los desplazamientos de los punteros, el del origen (*OffFrom*) a cero, y el del destino (*OffTo*) lo calculamos como ya conocemos: *x+(ANCHO*y)*.

Dos bucles encadenados y una comparación se encargan de escribir convenientemente el carácter. Los bucles actualizan, en su definición, las variables *OffFrom* y *OffTo* y la comparación comprueba si el valor es transparente (0) o no:

```
pFonts = Font.BitmapRGB+LettOff;
OffFrom = 0;

OffTo  = x+(y*VideoMode.hres);
for
(CntH=0;CntH<Height;CntH++,Off
To+=VideoMode.hres-Wide,
OffFrom+=Font.hres-Wide)
  for
(CntW=0;CntW<Wide;CntW++,Off
To++,OffFrom++)
    if (pFonts[OffFrom]!=0)
Where[OffTo]=pFonts[OffFrom];
```

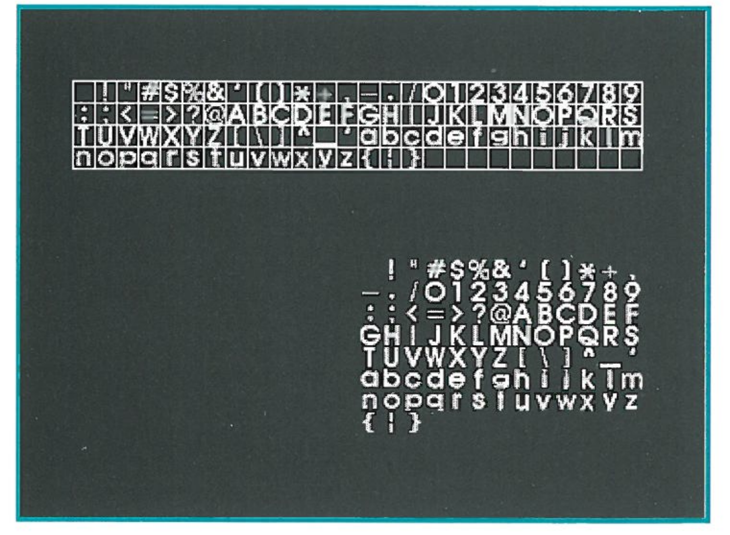
CFont::write. Ésta lee la cadena de texto que se pretende escribir y llama a la función *PutLetter* con cada carácter de la cadena, pasando a dicha función los valores adecuados, esto es, el desplazamiento donde se encuentra la definición de la fuente. El proceso se repite hasta que se encuentra el carácter NULL, que indica el final de la cadena de caracteres.

CFont::End. Libera la memoria del fichero PCX donde se encuentra la definición de la fuente.

EL EJEMPLO

El ejemplo que acompaña a este artículo ilustra el sencillo manejo de la clase. Destacar, únicamente, la inclusión de un fichero gráfico, que se emplea para ilustrar el funcionamiento del *clipping*.

FIGURA 2.

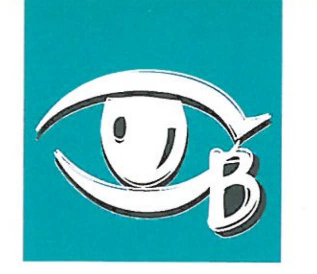


VISUAL BASIC

Nivel: Medio

Capítulo 12º

Autores: Juan Manuel y Luis Martín



Programación orientada a objetos

La programación Orientada a Objetos es la metodología de programación más utilizada hoy en día por los lenguajes de alto nivel. Visual Basic incorpora la capacidad de trabajar con objetos de una forma cómoda y sencilla para el programador. De hecho, todos los elementos que intervienen en la creación de una aplicación con Visual Basic deben entenderse como objetos.

Todas las aplicaciones desarrolladas con Visual Basic trabajan con unas entidades denominadas objetos. Los objetos pueden ser definidos como combinaciones de datos y código que son tratados como una sola unidad. A lo largo de los artículos anteriores de este curso se ha estado trabajando con objetos, ya que tanto los controles como los propios formularios son objetos. Sin embargo, dentro de una aplicación desarrollada con Visual Basic pueden identificarse muchos otros objetos, como por ejemplo, una base de datos o un gráfico.
Además, Visual Basic permite trabajar con los objetos como si se tratase de otro tipo de datos, permitiendo crear variables que contienen objetos para manipularlos. También es posible utilizar objetos provenientes de otras aplicaciones externas. Para ello, los objetos poseen una característica, denominada *Automatización*, que forma parte del denominado *Modelo de Objetos Componentes* (COM). Esta tecnología permite estandarizar la exportación de objetos entre diferentes aplicaciones para que éstos puedan ser reutilizados.

MANEJO DE OBJETOS

Como ya se ha mencionado anteriormente, tanto los formularios como los controles son objetos. En general, los objetos usados por Visual Basic se componen de propiedades, métodos y eventos. Las propiedades son datos que indican las características y atributos particulares del objeto. Los métodos son las operaciones que pueden realizarse con los objetos. Por último, los eventos son aquellos sucesos susceptibles de ser detectados por el objeto.
El manejo de las propiedades dentro de una aplicación es similar al de las variables. Para asignar un valor a una propiedad de un objeto será necesario indicar, además del nombre de la propiedad, el nombre del objeto, separando ambos por un punto. Por tanto, la sintaxis general será la siguiente:

Objeto.Propiedad = Expresión

Cada una de las propiedades de un objeto admite valores de un tipo de datos concreto (*Integer*, *String*, *Boolean*, etc).

Form1.Caption = "Aplicación de ejemplo"

Del mismo que es posible asignar un valor a una propiedad, también es posible obtener el valor almacenado en la misma. Para ello, bastará hacer referencia a la propiedad utilizando una sintaxis similar a la de asignación, que puede expresarse de la siguiente forma:

Variable = Objeto.Propiedad

En este otro caso, también es importante tener en cuenta el tipo de datos almacenado en la propiedad.

```
If Len(Form1.Caption) > 15 Then _
Print Upper(Labe1.Caption & Text2.Text)
```

Cuando lo que se desea hacer es ejecutar un método de un objeto, la sintaxis también es similar. Así, debe especificarse el nombre del objeto y del método, separados por un punto. Sin embargo, en este caso existen algunas variantes, dependiendo de si el método es una subrutina o una función, y de si admite o no argumentos.
Si el método no devuelve ningún valor, es decir, si se trata de una subrutina, puede expresarse de la forma siguiente:

Objeto.Método [ListaArgumentos]
Cuando el método va acompañado de argumentos, éstos se especifican separados entre sí por comas.

Objetos, clases e instancias

El trabajo con objetos proviene de una metodología de diseño de aplicaciones denominada Programación Orientada a Objetos. Así, cada una de las entidades que intervienen en una aplicación es un objeto. Dentro de los objetos, es posible diferenciar dos partes:
Datos: Son los atributos que definen las características propias del objeto. Generalmente, consisten en variables que contienen información relativa al objeto.
Acciones: Son las operaciones que el objeto puede realizar. Generalmente, consisten en procedimientos que realizan una acción relacionada con el objeto.
Dentro de la nomenclatura utilizada en Visual Basic, los datos de un objeto se denominan propiedades, y las acciones que puede realizar se denominan métodos. De esta forma, los objetos manejados por una aplicación incorporarán su propio repertorio tanto de propiedades como de métodos.
Todos aquellos objetos que sean de un mismo tipo serán creados a partir de una definición general denominada clase. Por tanto, las clases son una definición general de un tipo de objetos. Además, cada uno de los objetos creados a partir de una clase se denomina instancia.
Estos conceptos pueden aclararse si se compara a los objetos con variables estándar. Así, el tipo de datos a partir del cual se crea cada variable representa la clase. De esta forma, podría decirse que las clases incorporadas por Visual Basic serían Integer, String, Boolean, etc. Por otra parte, cada una de las variables declaradas a partir de estos tipos de datos representarán las instancias (Contador, CadenaTexto, Actualizado, etc).
Los objetos disponibles en el cuadro de herramientas de Visual Basic son las distintas clases de los objetos control. Al añadir un nuevo control a un formulario, lo que en realidad se está haciendo es crear una nueva instancia de dicha clase. De esta forma, las clases son CommandButton, Label o Timer, mientras que Command1, Label1 o Timer1 serían instancias de dichas clases.
Por el contrario, cada formulario creado en tiempo de diseño constituye una clase distinta. Así, cuando se pasa a tiempo de ejecución, cada uno de los formularios será una instancia a partir de su propia clase de formulario. En general, puede considerarse que un módulo de formulario es un módulo de clase con interfaz visible.



El examinador de objetos

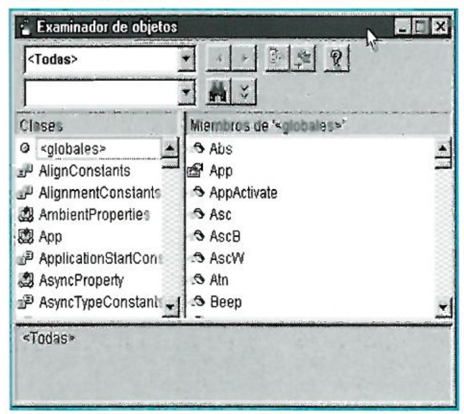
Visual Basic dispone de una herramienta, denominada Examinador de objetos, que muestra las clases disponibles para la utilización de objetos dentro de una aplicación. Dichas clases pueden proceder tanto de las librerías de objetos de Visual Basic, como de otras librerías externas, o de los distintos módulos que forman parte del proyecto. De esta forma, es posible visualizar las propiedades y los métodos de todos lo objetos. Para acceder a la misma, debe seleccionarse la opción Examinador de objetos del menú Ver (véase la figura 1).
La parte superior muestra la lista desplegable Bibliotecas/Proyectos, que permite seleccionarse las distintas librerías de objetos disponibles para la aplicación.
Es posible añadir y eliminar librerías a esta lista utilizando para ello el cuadro de diálogo de Referencias. Este cuadro de diálogo también permite añadir otros proyectos de Visual Basic con el fin de visualizar sus módulos, clases, métodos y propiedades. Su aspecto se muestra en la figura 2.
Al seleccionar una determinada librería, la lista Clases/Módulos muestra los nombres de las clases disponibles en la misma. Igualmente, al seleccionar una clase en esta última, la lista Metodos/Propiedades muestra las propiedades y los métodos de dicha clase.
Además, dentro de algunas librerías aparecen definidas una serie de constantes intrínsecas que también pueden mostrarse con el examinador de objetos. Así, por ejemplo, en la librería VB, la enumeración vbAlignConstants muestra los valores vbAlignBottom, vbAlignLeft, etc.

Command1.SetFocus
Command1.Move 1200, 750

Por el contrario, cuando el método es una función, es decir, devuelve un valor, los argumentos deben encerrarse entre paréntesis.

Anchura = Form1.TextWidth(Cadena)

FIGURA 1. ASPECTO DEL EXAMINADOR DE OBJETOS.



VARIABLES OBJETO

Las variables de Visual Basic, además de almacenar valores de los distintos tipos de datos, también permiten almacenar objetos. A este tipo de variables se les denomina *variables objeto*. Tal y como ocurre con el resto de variables, para utilizar variables objeto es necesario declararlas primero. La declaración se realiza de forma similar al resto de variables, utilizando para ello las mismas sentencias. La sintaxis general puede ser expresada de la siguiente forma:

{Dim | ReDim | Static | Private | Public} Variable As [New] Clase

En el argumento *Clase* debe indicarse el tipo de datos de la variable, que debe ser el nombre de una clase. Una vez declarada la variable objeto, ésta sólo podrá hacer referencia a objetos pertenecientes a la clase correspondiente.
Es posible declarar variables objeto que hagan referencia a un tipo de control específico, como *TextBox*, *ListBox*, *Label*, etc. Igualmente, es posible declarar variables objeto que hagan referencia a un tipo específico de formulario de la aplicación. A este tipo de variables de les denomina variables objeto *específicas*.

Dim Boton As CommandButton
Dim Frm1 As New Form1

De esta forma, la variable *Boton* puede hacer referencia a cualquier botón de comando de la aplicación. Por su parte, la variable *Frm1* sólo podrá hacer referencia a instancias del formulario *Form1*. Si la aplicación contiene otros formularios, la variable no podrá hacer referencia a ellos.
Sin embargo, existen clases que permiten declarar variables objeto que hacen referencia a cualquier tipo de control, a cualquier tipo de formulario, e incluso a cualquier tipo de objeto. A este tipo de variables se les denomina variables objeto *genéricas*.

Dim Cnt As Control
Dim Frm As Form
Dim Obj As Object

En Visual Basic hay cuatro tipos de objetos genéricos, que permiten declarar variables para hacer referencia a distintos tipos de objetos:
Form: Puede hacer referencia a cualquier formulario de la aplicación, incluido el formularios MDI.
Control: Puede hacer referencia a cualquier tipo de control disponible en la aplicación.
MDIForm: Sólo puede hacer referencia al formulario MDI.
Object: Puede hacer referencia a cualquier objeto de la aplicación, ya sea un formulario o un control.
Visual Basic permite declarar variables objeto que hagan referencia un formulario específico de la aplicación, pero no es posible declarar variables objeto que hagan referencia a un control específico. Así, por ejemplo, la siguiente declaración es incorrecta:

Dim Lb1 As Label1

Para que una variable objeto haga referencia a un objeto determinado, éste debe ser asignado a la variable. Esta operación se realiza mediante la sentencia *Set*, cuya sintaxis es la siguiente:

Set Variable = {[New] Objeto | Nothing}

En el argumento *Objeto* debe indicarse el nombre del objeto que se desea asignar, y debe ser del mismo tipo que se declaró para la variable. También es posible utilizar otra variable objeto del mismo tipo, o un objeto devuelto por un método o función.
Para eliminar la referencia de una variable a cualquier objeto existente, debe asignársele un valor especial. Se trata de la palabra reservada *Nothing*. Cuando se asigna este valor a una variable, ésta no hace referencia a ningún objeto.

Dim Lb As Label
Set Lb = Label1
Lb.Caption = "Hola"
Set Lb = Nothing

Las variables objeto también pueden ser utilizadas como parámetros en las llamadas a procedimientos. De esta forma, es posible pasar un objeto al procedimiento indicándolo simplemente en la lista de argumentos. Así, por ejemplo, el siguiente procedimiento reduce a la mitad el tamaño de un control.

```
Private Sub Mitad(Ctrl As Control)
  Ctrl.Width = Ctrl.Width / 2
  Ctrl.Height = Ctrl.Height / 2
End Sub
```

LA CLAUSULA NEW Y EL OPERADOR IS.

La cláusula *New* permite crear nuevas instancias de una determinada clase en tiempo de ejecución. Esta cláusula puede ser incluida tanto en la sentencia de declaración de la variable objeto, como en una sentencia de asignación de la misma. Esta cláusula sólo puede ser utilizada con formularios específicos de la aplicación, clases definidas en los módulos de clase, colecciones de objetos y objetos de automatización OLE.



Cuando se utiliza la cláusula New en una sentencia de asignación *Set*, se crea automáticamente una nueva instancia de la clase, que será asignada a la variable especificada. Así, por ejemplo, para crear una nueva instancia del formulario por defecto y visualizarla, puede utilizarse lo siguiente:

```
Dim Frm As Form1
Set Frm = New Form1
Frm.Show
```

Cuando se crean instancias de un formulario, éstas no se visualizan inicialmente, ya que tienen su propiedad *Visible* a *False*. Además, en el caso de los formularios no basta con asignar *Nothing* a la variable para eliminar el formulario, sino que debe descargarse explícitamente con la sentencia *Unload*.
La cláusula *New* no puede ser utilizarse con variables objeto de tipo genérico, ni con variables objeto de tipo control específico.
El ejemplo 1 del CD-ROM ilustra la creación de múltiples instancias de un formulario mediante la utilización de la cláusula *New*.
Para comparar variables objeto, Visual Basic dispone del operador *Is*. Se trata de un operador especial que devuelve *True* cuando ambas hacen referencia al mismo objeto y *False* en caso contrario.

```
Set Obj1 = Text1
Set Obj2 = Text1
Set Obj3 = Text2
Print Obj1 Is Obj2          'Resultado: True
Print Obj2 Is Obj3          'Resultado: False
```

La utilización de este operador en combinación con la palabra reservada *TypeOf*, permite realizar comprobaciones del tipo de objeto. Su sintaxis es la siguiente:

```
TypeOf VarObjeto Is TipoObjeto
```

En el argumento *VarObjeto* debe especificarse el nombre de una variable o referencia a un objeto, mientras que en *TipoObjeto* debe especificarse cualquiera de los tipos de objetos de la aplicación, ya sea genérico o específico. Este tipo de construcción sólo puede ser utilizado como condición de una sentencia de control *If...Then...Else*.

```
If TypeOf Obj Is TextBox Then
  Print "Se refiere a un cuadro de texto"
End If
```

COLECCIONES DE OBJETOS

Los objetos de las aplicaciones realizadas con Visual Basic se organizan en jerarquías, de forma que puede considerarse que unos objetos están contenidos en otros, que a su vez están contenidos en otros, y así sucesivamente. Para hacer referencia en el código a un objeto que se encuentra contenido en otro, es necesario indicar el nombre de ambos, separados por un punto. Así, por ejemplo, para asignar un valor a la propiedad *Name* del objeto *Font* de un control *Text1* debe utilizarse la siguiente sentencia:

```
Form1.Text1.Font.Name = "Courier New"
```

Visual Basic dispone de un tipo especial de objetos que permite realizar agrupaciones de otros objetos. Se trata de los objetos colección (*Collection*), que están constituidos a su vez por un cierto número de objetos ordenados. Cada uno de los objetos de la colección se denomina *miembro*. La única relación que existe entre los miembros de una determinada colección es que pertenecen a ésta, no siendo necesario que sean del mismo tipo.
Para crear una colección se utilizan las mismas sentencias de declaración de variables, aunque especificando como tipo de datos la palabra reservada *Collection*.

```
Dim Grupo As New Collection
```

Dentro de un objeto *Collection*, los miembros se sitúan ordenadamente, de forma similar a como lo hacen los elementos de una matriz. Así, a cada objeto le corresponde un número de índice que indica su posición dentro de la colección. De esta forma, es posible hacer referencia a un miembro indicando el nombre de la colección, seguido del valor del índice encerrado entre paréntesis.
También es posible hacer referencia a un determinado miembro, especificando una clave de identificación en lugar del índice. Esta clave se indica cuando se añade el miembro a la colección, y debe expresarse como una cadena de caracteres. Finalmente, es posible indicar la clave del miembro literalmente, separándola del nombre de la colección mediante el carácter admiración (!).

```
Grupo(2).Width = 1280
Grupo("T2"). Width = 1280
Grupo!T2. Width = 1280
```

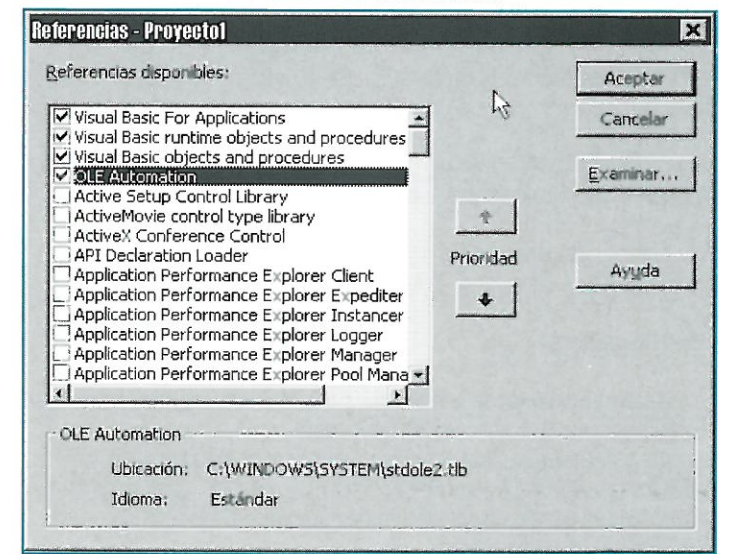
Los objetos *Collection* disponen de una única propiedad. Se trata de la propiedad *Count*, que devuelve el número total de miembros que contiene la colección. De esta forma, es posible recorrer todos los miembros de una colección a través de su índice mediante un bucle For...Next.

```
For N = 1 To Grupo.Count
  'Operación con el miembro N-ésimo
Next
```

Una vez creado un objeto *Collection*, es posible añadirle nuevos miembros utilizando para ello el método *Add*. La sintaxis de este método es la siguiente:

```
Objeto.Add(Miembro[, Clave[, Antes[, Después]]])
```

FIGURA 2. ASPECTO DEL CUADRO DE DIALOGO REFERENCIAS.



Contenedores de controles

Visual Basic dispone de varios controles dentro de los cuales pueden situarse otros controles, es decir, controles capaces de contener a otros. A este tipo de controles se les denomina contenedores, y su utilización también constituye una jerarquía de objetos.
Dentro de los controles estándar, los contenedores son los marcos (Frame) y los cuadros de dibujo (PictureBox), además de los propios formularios. Así, por ejemplo, es posible situar un botón de comando en un cuadro de dibujo que se encuentra dentro de un marco que, a su vez, se encuentra dentro del formulario.
En tiempo de ejecución es posible acceder al contenedor de un control a través de sus propiedad Container, que representa el objeto contenedor. También es posible cambiarlo dinámicamente de contenedor, asignando un nuevo control a la propiedad Container con la sentencia Set. El ejemplo 2 del CD-ROM ilustra este tipo de operaciones.

En el argumento *Miembro* debe indicase el nombre del objeto que se desea agregar a la colección. Por su parte, el argumento *Clave* permite especificar una cadena de caracteres que identificará al miembro cuando se desee acceder a él.

```
Dim Grupo As New Collection
Grupo.Add Text2, "T2"
```

Por defecto, cada vez que se añade un nuevo miembro a una colección, éste se sitúa al final de la misma. Sin embargo, es posible especificar la posición donde se desea insertar el miembro, utilizando para ello los argumentos *Antes* y *Después*. Estos argumentos son mutuamente excluyentes, es decir, si se especifica uno de ellos no puede especificarse el otro.
En el argumento *Antes* puede especificarse una posición de la colección entre 1 y *Count*. De esta forma, el nuevo miembro se situará delante del miembro indicado por este argumento. También es posible indicar una clave de identificación en vez del índice posicional. El argumento *Después* es muy similar, aunque el nuevo miembro será insertado detrás del miembro especificado.<

```
Grupo.Add (Label1,"L1","L2")
```

Para eliminar un miembro de una colección se utiliza el método *Remove*, cuya sintaxis es la siguiente:

```
Objeto.Remove(Índice)
```

En el argumento *Índice* debe especificarse la posición del miembro que se desea eliminar, o su clave de identificación.

```
Grupo.Remove ("L1")
Grupo.Remove (Grupo.Count)
```

Para obtener una referencia a un miembro determinado de una colección puede utilizarse el método *Item*, cuya sintaxis es la siguiente:

```
Objeto.Item(Índice)
```

En el argumento *Índice* debe especificarse la posición del miembro al que se desea acceder o su clave de identificación.
La sentencia *For Each...Next* también puede ser utilizada para recorrer todos los miembros de una colección. Así, por ejemplo, para imprimir la propiedad *Caption* de una colección de etiquetas puede utilizarse el siguiente fragmento de código:

```
For Each Elemento In Etiquetas
  Print Elemento.Caption
Next
```

LAS COLECCIONES PREDEFINIDAS

Visual Basic incorpora una serie de colecciones de objetos predefinidas. De esta forma, es posible manejar los formularios de una aplicación, los controles de un formulario o las impresoras disponibles del sistema, mediante el uso de una colección de objetos.
La colección *Forms* contiene todos los formularios de la aplicación que se encuentran cargados. Así, por ejemplo, para ocultar todos los formularios cargados puede utilizarse lo siguiente:

```
For Each Frm In Forms
  Frm.Visible = False
Next
```

La colección *Controls* contiene todos los controles de un formulario. Se trata de una variable intrínseca a nivel de formulario que permite acceder a todos los controles del formulario. Así, por ejemplo, para deshabilitar todos los botones de comando de un formulario, puede utilizarse el siguiente fragmento de código:

```
For Each Ctrl In Controls
  If TypeOf Ctrl Is CommandButton Then
    Ctrl.Enabled = False
  End If
Next
```

Por último, la colección *Printers* contiene todas las impresoras disponibles del sistema. Se trata de una variable global intrínseca, que permite acceder a cada una de las impresoras del sistema.

En Visual Basic hay cuatro tipos de objetos genéricos: Form, Control, MDIForm y Object

Así, por ejemplo, para establecer como impresora predeterminada la primera impresora capaz de imprimir en color, puede utilizarse el siguiente fragmento de código:

```
For Each Prn In Printers
  If Prn.ColorMode = vbPRCMColor Then
    Set Printer = Prn
    Exit For
  End If
Next
```

CREACION DE CLASES PROPIAS

Visual Basic permite al programador crear sus propias clases. Para ello, se utiliza un tipo de módulos, denominados *módulos de clase* (*Class module*). Se trata de módulos muy similares a los de formulario. La principal diferencia entre ambos es que los módulos de formulario presentan una interfaz visible, mientras que los de clase no.
Cada módulo de clase incluido en el proyecto sólo permite definir una única clase. Las propiedades de dicha clase deben ser definidas mediante la utilización de variables a nivel de módulo o procedimientos de propiedades. Los métodos de la clase deben ser definidos mediante procedimientos *Sub* y *Function*, según devuelvan o no un valor. Los objetos o instancias sólo podrán ser creados a partir de la clase en tiempo de ejecución.



Para crear un nuevo módulo de clase debe seleccionarse la opción *Agregar Módulo de Clase* del menú *Proyecto*. Según se van añadiendo nuevos módulos de clase, Visual Basic les asigna los nombres *Class1*, *Class2*, etc. Los módulos de clase aparecen dentro del Explorador de proyectos en una carpeta denominada *Módulos de clase*. Cada módulo de clase dispone de tres propiedades predefinidas:
Name: Contiene el nombre de la clase en forma de cadena de caracteres. Inicialmente coincide con el nombre del módulo.
Public: Contiene un valor lógico que indica si la clase será accesible desde otras aplicaciones fuera del proyecto.

Cada módulo de clase incluido en el proyecto sólo permite definir una única clase

Instancing: Contiene un valor numérico que indica la forma en que puede accederse a la clase desde otras aplicaciones cuando la propiedad *Public* se encuentra a *True*.
En la ventana de propiedades sólo se mostrará la primera de estas tres propiedades. Las dos últimas están muy relacionadas con la creación de controles ActiveX, por lo que serán analizadas en posteriores artículos.
La propiedad *Name* determina el nombre de la clase, y su valor será el que se utilice a la hora de declarar objetos de dicha clase. Para ello, se utilizan las mismas sentencias que en la creación de instancias de formularios, incluyendo también la palabra reservada *New*. La sintaxis para la creación de instancias de una clase en tiempo de ejecución puede expresarse de la siguiente forma:

```
Dim Variable As New Clase
```

Una vez creado un nuevo módulo de clase en el proyecto, es posible añadir propiedades al mismo. La forma mas sencilla de añadir propiedades a una clase consiste en declarar variables públicas a nivel de módulo. Para ello, debe accederse a la sección de declaraciones dentro de la Ventana de código del módulo de clase.
La declaración de las propiedades con la sentencia *Public* permitirá acceder a éstas desde otros módulos cuando se haya creado una instancia de la clase. Esto también incluye a la propiedad *Name* del objeto, que aunque ya se encuentra definida, debe ser declarada siempre que vaya a ser utilizada desde otro módulo. Para ello, debe incluirse la siguiente instrucción en la sección de declaraciones del módulo de clase:

```
Public Name As String
```

También es posible añadir nuevas propiedades a la clase mediante la creación de procedimientos de propiedades. Para ello, lo primero que debe hacerse es crear en la zona de declaraciones del módulo de clase una variable privada que contendrá el valor de la propiedad.

```
Private mdcValor As Integer
```

Desde la ventana de código debe seleccionarse la opción *Agregar procedimiento* del menú *Herramientas*. En dicho cuadro de diálogo habrá que indicar el nombre del procedimiento, que en este caso coincide con el nombre de la propiedad, el tipo de procedimiento (que será *Propiedad*) y el alcance del mismo (que será *Público*).
Tras ello, en la ventana de código aparecen dos nuevos procedimientos. El procedimiento *Let* se ejecuta siempre que se asigne una nueva cantidad a la propiedad. Por ello, en el cuerpo de dicho procedimiento lo que habrá que hacer es asignar el nuevo valor a la propiedad.

```
Public Property Let Valor(ByVal vNewValue As Integer)
  mdcValor = vNewValue
End Property
```

Por su parte, el procedimiento *Get* se ejecuta cada vez que se recupera el valor de la propiedad. Por lo tanto, lo único que habrá que hacer en este procedimiento de evento es devolver el valor almacenado en la variable privada.

```
Public Property Get Valor() As Integer
  Valor = mdcValue
End Property
```

La utilización de procedimientos de propiedad en vez de variables públicas en un módulo de clase permite crear propiedades de sólo lectura para dicha clase. Para ello, pueden declararse variables privadas, que quedarán ocultas a otros módulos, definiendo también un procedimiento de propiedad que devuelva su valor de forma controlada. Además, las propiedades definidas mediante procedimientos de propiedad pueden tener asociada una descripción y un tema de ayuda interactiva en el examinador de objetos.
La creación de métodos en un módulo de clase se realiza mediante la definición de subrutinas y funciones públicas. Estos métodos podrán ser utilizados con los objetos creados en tiempo de ejecución, y permitirán acceder desde otros módulos a las propiedades de la clase que se encuentran ocultas.

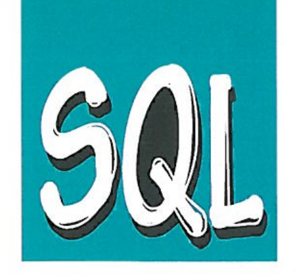
Los métodos que devuelven valores deben ser definidos como funciones, mientras que los que no devuelven ningún valor pueden ser definidos como subrutinas. Además, es posible definir métodos privados que sólo podrán ser llamados desde los procedimientos del propio módulo de clase.

Los objetos de las aplicaciones realizadas con Visual Basic se organizan en jerarquías

Los módulos de clase disponen además de dos procedimientos de evento predefinidos. Se trata de los eventos *Initialize* y *Terminate*, que se producen cuando se crea o destruye una nueva instancia de la clase, respectivamente.
El evento *Initialize* se produce cuando se crea un objeto a partir de la clase.
Para ello, bastará con hacer referencia a alguna propiedad del mismo, después de haber sido declarado o asignado a una variable con la palabra clave *New*. Este evento también se produce cuando se crea un nuevo objeto utilizando la función *CreateObject*.
El evento *Terminate* se produce cuando se destruye el objeto, es decir, cuando todas las variables que hacen referencia a él se ponen a *Nothing* o quedan fuera del ámbito de visibilidad.
Los eventos *Initialize* y *Terminate* también son aplicables a las nuevas instancias creadas a partir de formularios estándar y formularios MDI. En este caso, el evento *Initialize* se ejecuta antes del evento *Load*, mientras que *Terminate* se ejecuta después de *Unload*.
También es posible definir eventos dentro de una clase. Sin embargo, dado que las clases carecen de interfaz de usuario, no parece muy apropiada la adición de esta posibilidad dentro de un módulo de clase. Esta utilidad si resultará interesante dentro de la creación de controles ActiveX, por lo que será analizada con mayor profundidad en artículos posteriores, en los que se explicará de forma detallada el proceso de creación de controles ActiveX por parte del usuario.

Conclusión

En este artículo se ha tratado de dar una visión general de las posibilidades incorporadas por Visual Basic en lo referente a la Programación Orientada a Objetos. En posteriores artículos de carácter más avanzado se analizará de forma mucho más clara la utilidad de la creación e inclusión de clases propias dentro de un proyecto.



Las DBTools.h++ [II]

En la pasada entrega se estudiaron las bases de las tan usadas DBTools.h, en este número se verán los aspectos restantes de esta herramienta.

Y así, se vera el modelo que usan las DBTools para tratar los errores, además de las técnicas de manipulación de los mismos, para llegar en el plano central de este artículo en lo que básicamente esta centrado la importancia de este tema, las sentencias de selección embebidas.
Por último se tratarán como tema ya secundario la inserción y el borrado a través de las DBTools.h++.

EL MODELO DE ERRORES EN LAS DBTOOLS

Las Dbtools, dividen en dos las categorías de los errores, siendo estos errores internos y externos.

- Los errores internos son debidos a la lógica del programa, son errores como el mal control del límite de una tabla, inserción de un puntero nulo, un mal uso de los datos, etc. , en los primeros dos casos (errores con el límite de una tabla, y inserción de punteros nulos) son errores difíciles de detectar y bastante costosos, mientras que los otros son fáciles por lo que se pueden eliminar en la fase de construcción del software.
- Los errores externos son causados por condiciones externas como cuando la memoria se desborda, el servidor de bases de datos se cae, cuando se produce un error en la consulta, etc., por lo tanto no son fácilmente predecibles.

Para controlar los errores la DBTools tienen una clase llamada *RWDBStatus*, que se encarga de encapsular el estado de error de un objeto u operación. De tal manera que los objetos que interactuan con la base de datos contienen una instancia de esta clase.
Todos los miembros que tienen un estado de error también tienen una función miembro "status" que devolverá una copia del objeto estado. Esta función interactua con la base de datos pero no cambia el estado del objeto, sino que devuelve una instancia de *RWDBStatus*.
Todas los objetos que tienen un estado también tienen una función miembro "isValid", que devolverá un valor lógico basado en el estado de este.
RWDBStatus contiene un numero que indica el estado del objeto, pudiendo valer éste un error Server Error, Vendor Library Error y ok, este número le podemos obtener con la función *ErrorCode*, pudiendo este tomar el siguiente valor:
ErrorCode errorCode() const;

Las DBTols también ofrece el texto del código de error con la siguiente función miembro:
RWCString message() const;
Los códigos de error y mensajes tanto de las librerías de clientes como las del servidor son también capturadas gracias a las funciones miembro:
Long vendorError1() const;
Long vendorError2() const;
RWCString vendorMessage1() const;
RWCString vendorMessage2() const;

Veamos a continuación un código ejemplo de un manejador de errores:

```
Void ManejadorErrores (const RWDStatus& s)
{
        if (s.errorCode() != RWDBStatus::ok){
        crr << s.errorCode()<<" "<<s.message();
                if (s.errorCode()==RWDBStatus::severErrorll
            s.errorCode()==RWDBStatus::vendorLib)
                    cerr<<": "<<s.vendorError1()<s.vendorMessage1();
                    cerr<<end1;
                    exit(-1);}
}
```

TECNICAS DE MANIPULACION DE ERRORES

Existen varias técnicas en la manipulación de los errores, entre estas son:

- Ignorar el error, esto se realiza de la siguiente manera. El productor propaga el estado de su error a sus productos (en el paradigma productor/consumidor), de tal manera que un objeto con un estado erróneo es un estado no funcional, y por lo tanto este objeto será ignorado. Esto se verá más claro en el siguiente código:

```
//...
RWDBDatabase aDB = RWDBManager::database (
                "SYBASE", "milogin", "mipassword", "nom_serv");
RWDBTable aTable = aDB.table("TheTable");
RWDBReader aReader = aDB.reader();
Int x;
While (aReader()){
AReader>>x;
ProcessData (x);}//...
```

- Chequeo del estado de objetos. El chequeo del estado de los objetos es muy simple y permite un fácil manejo de los errores y una localización local de estos, esta técnica se puede ver en el siguiente código:

```
//...
RWDBDatabase aDB = RWDBManager::database (
                "SYBASE", "milogin", "mipassword", "nom_serv");
if (!aDB.isValid()) outputError(aDB.status());
RWDBTable aTable = aDB.Table("TheTable");
RWDBReader aReader = aTable.reader();
If (¡!areader.isValid())
OutputError(aReader.status());
Int x;
While (aReader()){
AReader >> x;
ProcessData (x);}//...
```

- Otro forma de manipular los errores, es construyendo un manejador de errores. Cada instancia de *RWDBStatus* contiene un puntero a una función conocida como un manejador de errores. Cuando sucede un evento que cambia el estado de una instancia de *RWDBStatus*, el manejador de errores es llamado con la instancia de este objeto como argumento. Por defecto no existe ningún manejador de errores. Para instalar un nuevo manejador debe ser usada la función seterrorhandler de *RWDBStatus*, para declarar el manejador debe hacerse de la siguiente manera:
Void mimanejadordeerrores (cons RWDBStatus&);



A continuación se muestra un código ejemplo de lo visto en este apartado:

```
Void mimanejadordeerrores (cons RWDBStatus&);
{
if (s.errorCode() ¡= RWDBStatus::ok){
                    cerr<<s.errorCode() << ""<< s.message();
                    if(s.errorcode()==RWDBStatus::severErrorll
   s.errorCode()==RWDBStatus::vendorLib)
                    cerr<< ":"<<s.vndorError1()<<s.vendorMess1();
cerr<<end1;} }
```

- Por último se puede utilizar el manejador de excepciones del C++. Para ello *RWDBStatus* tiene una función miembro llamada *raise*(),siendo su especificación la siguiente:

```
Void throwExceptionOnError(const RWDBStatus& s){
s.raise();}
```

A continuación se mostrará un ejemplo que utilice el manejador de excepciones del C++.

```
//...
RWDBManager::setErrorHandler (throwExceptionOnError);
Try {
RWDBDatabase aDB=RWDBManager::database   (SYBASE,"milogin",
"mipassword", "NOM_SERVER");}
catch (Xmsg x){
cout<<"Error abriendo la base de datos"<<x.msg()<<end1;}//...
```

SELECCION DE DATOS EN LAS DBTOOLS, EL SELECT EMBEBIDO

En ejemplos puestos en la anterior entrega, fueron recuperados todos los datos de una tabla, pero generalmente cuando accedemos a una base de datos, nosotros queremos obtener únicamente parte de esta información.

Por otro lado como se vio en las primeras entregas, en SQL existe una orden para recuperar parte de los datos de una base de datos, así como datos de varias tablas relacionadas, como han podido darse cuenta ya la mayoría de nuestros lectores, se trata de la orden Select.

Vamos a recordar algunas Select, para ubicar a nuestros lectores.

Dadas las tablas 1 y 2, se requiere seleccionar el nombre de los estados y las ciudades, cuyas poblaciones sean mayores un millón de habitantes.

```
SELECT S.nombre, C.nombre
FROM Estados S, Ciudades C
WHERE S.poblacion>1000000
Order by 1
```

En las DBTools.h++, la acción de selección de datos ha sido encapsulada dentro de un objeto. El *RWDBSelector* es otra encapsulameinto del DML.

El *RWDBSelector* es una herramienta utilizada para describir la información que será recuperada dentro de la base de datos. Como ya se explico en anteriores entregas, un selector puede restringir una consulta por elección de solo unas columnas a través de un criterio especificado.

Las Instancias de *RWDBSelector* son producidas por *RWDBDatabases*.

Cada uno de los selectores producidos deben ser inicializados por los parámetros que describen las consultas.

Los parámetros en una orden Select están compuestos por las diferentes cláusulas de la orden. Una instancia *RWDBSelector* tiene un conjunto de funciones miembro para actualizar dichos parámetros. Pongamos como referencia la orden Select anterior

```
SELECT S.nombre, C.nombre
FROM Estados S, Ciudades C
WHERE S.poblacion>1000000
Order by 1
```

Como se puede ver esta típica orden Select consta de varias partes (lista de Select, cláusula where, cláusula from, cláusula order by).

La clase *RWDBSelector* especifica a través del operador de sobrecarga << expresiones de columnas, funciones y/o literales de la lista de select.

También tiene una función miembro where que acepta expresiones de columna.

Puede inferir automáticamente por referencias en la lista de select y la cláusula where o puede ser especificado a través de la cláusula from, es decir, si no es especifica la cláusula from buscará todas las tablas que contenga los campos especificados en la parte de la lista de select.

Por otro lado el *RWDBSelector* tiene una función miembro execute que devuelve el resultado en *RWDBResult*. El resultado de la consulta debe ser buscado en la tabla resultante.

Cuando la función miembro "execute" es invocada, el objeto *RWDBSelector* crea una cadena que representa el selector como una orden SQL SELECT.

La función miembro where de *RWDBSelector* acepta una instancia de RWDBCriterion como un parámetro. *RWDBCriterion* es una estructura de datos que representa un árbol sintáctico de expresiones booleanas que serán pasadas al servidor.

Las DBTools construye el árbol sintáctico dinámicamente de una expresión escrita en sintaxis de C++.

```
ASelector.where(
        Table2["Z"]<RWDate(4,7,90)&&table1["A"]==table2["X"]);
```

La cláusula where sobre es una expresión de tres columnas, una de datos, dos operaciones relacionales y un operador lógico.

```
ASelector.where(
        Table2["Z"]<RWDate(4,7,90)&& table1["A"]==table2["X"]);
```

El árbol sintáctico se construye automáticamente lanzando los operadores de sobrecarga.

Estas expresiones son representadas por instancias de la clase *RWDBExpr*.

Las expresiones booleanas se representan por instancias de *RWDBCriterion*, y las listas de expresiones de una clase *RWDBSelector* son realmente una lista de instancias de *RWDBExpr*.

TABLA 1

ID	NOMBRE	NUMERO
1	NEW YORK	4
2	OREGON	3
3	CALIFORNIA	5
4	MONTANA	2

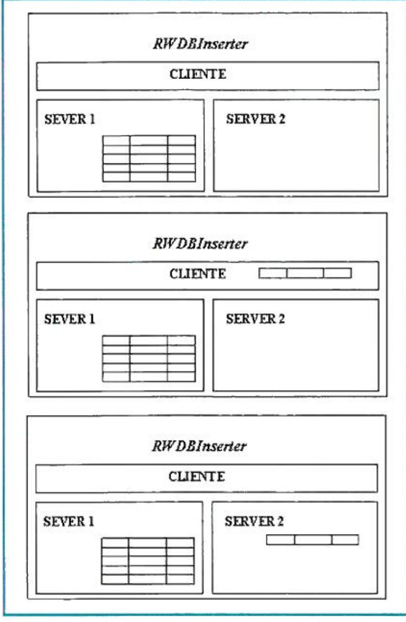
TABLA 2

ID	NOMBRE	NUMERO
1	BOISE	1
2	SALEM	2
3	OLIMPIA	3
4	WALLA	2

El compilador pasa todo a *RWDBExpr* y así ya aplica los operadores relacionales y donde sé *RWDBExpr*, que es lo que pide el where.
El SQL tiene un número de operaciones que no tiene el C++, estas funciones fueron implementadas como funciones miembros públicas de *RWDBColumn* y *RWDBExpr*, así pues las órdenes del SQL que el C++ no tiene son:
between, is null, like, match unique, left outer join, in.
De esta manera, y en el caso de *between* se implementaría de la siguiente manera:
ASelector.where(table2["X"]. between(2,4));
Esta función miembro de RWDBColumn y RWDBExpr devuelve una instancia de RWDBCriterion. Cuando se envía al servidor, esta expresión expande a una cláusula Between del SQL, según la siguiente... X *BETWEEN 2 AND 4...*

INSERCION DE DATOS

En SQL, los datos nuevos son insertados utilizando la orden INSERT. La orden INSERT de SQL puede implementarse de dos formas. La primera es insertar una única columna de datos dentro de una tabla. La segunda forma insertar múltiples columnas de datos de una fuente interna de la base de datos.
En el primer caso la orden de los valores debe coincidir exactamente a como están en la tabla. Solo se pueden insertar sobre tablas físicas de la Base de Datos. Un ejemplo de la primera forma son:
INSERT INTO TABLE VALUES (33,'Madrid',3.1415)
En el segundo caso tenemos podemos ver el siguiente formato:
INSERT INTO TABLE SELECT FROM TABLE1 WHERE A>2
La orden de SQL insert esta encapsuladas dentro de instancias de la clase *RWDBInserter.*
Los objetos *RWDBIserter* pueden ser usados en conjunción con instancias de *RWDBReader*, siendo el único requerimiento que los datos de las tablas utilizadas sean las mismas.
El operador << de *RWDBInserter* tiene una forma que acepta directamente el formato de la clase *RWDBReader.* Esto significa que cualquier lectura pueden tener salidas idénticas en la tabla destino según se muestra en las figuras 2, 3 y 4. En las cuales se puede ver que e copia cada una de las filas de la tabla origen y se pega en la tabla destino.
En la segunda forma de la orden INSERT de SQL se almacena dentro de la tabla la información recuperada de una orden Select de SQL. La forma de esta orden esta encapsulada por los objetos *RWDBInserter* y *RWDBSelector*, y tiene el siguiente formato:
RWDBInserter insercion=aTabla.inserter(aSelector);
De tal manera que el RWDBInserter es producido por una instancia RWDBTable con un objeto de selección que se pasa como parámetro, y todas las filas que den como resultado la cláusula select serán insertadas de una sola vez.
La inserción es realmente llevada a cabo cuando es llamada la función miembro execute, pudiendo tener esta el siguiente aspecto:
RWDBInserter insercion = aTabla.inserter (aSelector);
RWDBResult resultSet = insercion.execute();
Otra forma de insertar valores en una tabla es mediante el uso del operador operator<<, así pues este operador inserta columnas nuevas dentro de una tabla y tiene un formato semejante al que se muestra a continuación:
RWDBInserter& operator<< (lostOfTypes);
También se puede utilizar este operador para insertar valores nulos en una tabla como se muestra en el siguiente ejemplo:
RWDBInserter& operator<< (RWDBValueManip&);
que equivale a la siguiente expresión:
anInserter << NULL;



FIGURAS 2, 3 Y 4. PROCESO DE INSERTADO DE UN SERVIDOR A OTRO.

También este operador permite insertar columnas enteras dentro de una tabla de una fuente que proviene de una instancia a una lectura, pero para ello tanto los datos de entrada en la tabla como los datos de la tabla fuente deben ser compatibles, esta opción tiene el siguiente formato:
RWDBInserter& operator<< (RWDBReader&);
La pega que tiene este operador es que cada vez que se quiere insertar se ha de poner una línea de código. Para evitar este problema tenemos el operador operator[] el cual permite estar dentro de un orden de rango, a continuación se puede ver el siguiente ejemplo:

AnInserter["columna"]<<342;

EL BORRADO DE DATOS

Para borrar una columna de una tabla el SQL se sirve de la orden DELETE, la cual consiste en solo dos partes. La primera parte es solo el nombre de la tabla de la que se borrarán las columnas, la segunda parte es una cláusula WHERE que especifica el criterio para las filas que vamos a cerrar. Así pues una orden típica de borrado es la siguiente:
DELETE FROM TABLA1 WHERE NUM>15
Esta orden borra todas las filas que satisface la condición estipulada por la parte de WHERE.
Las DBTools.h usa esta acción de borrado de una forma encapsulada a través de instancias de la clase *RWDBDeleter* que son herramientas para borrado de filas de tabla no queridas en estas.
El objeto *RWDBDeleter* es producto de *RWDBTables*, y por supuesto una instancia de esta debe representar una tabla de la base de datos.
A través del uso de *RWDBResult*, es posible determinar como fueron borradas la columnas. La función miembro *rowCount* de *RWDBResult* devolverá el número de filas que han sido borradas.
RWDBDeleter es una función miembro pública que devuelve el elemento borrado como una orden de borrado de SQL en el dialecto apropiado, y esta será la misma cadena que devolverá execute al servidor de base de datos.
Cuando se realiza la orden de borrado a través del execute en el servidor de bases de datos, este devolverá el resultado a través de una instancia *RWDBResult.* Este objeto resultado puede ser consultado por una orden de borrado.
Existe una función miembro isvalid, que chequea el buen estado de la instancia *RWDBDeleter.* Si la orden de borrado ha fallado, chequea el estado de la instancia de *RWDBResult*, para ver en que situación se encuentra.
Status(), es una función miembro que devuelve una instancia *RWDBStatus* del objeto borrado, esta instancia puede ser utilizado para determinar el estado del error en el que se encuentra el objeto. La función miembro *table*() devuelve la tabla que originó la orden de borrado.
La función miembro where actualiza la cláusula de la orden delete.

Con
el procesador PENTIUM®II
seguimos por delante



Comelta, sa.



El nuevo procesador

PENTIUM®II

combina las más altas tecnologías del momento,

"poniendo en manos del usuario la más elevada productividad".

Los ordenadores de la serie **"QUASAR"** de **Comelta** incluyen **procesador PENTIUM II** a 266 Mhz, convirtiéndose así en las mejores y más potentes estaciones de trabajo.

MMX™ intel inside pentium®II PROCESSOR


Sí, deseo recibir más información sobre la serie QUASAR de ordenadores personales COP Comelta.
(Att. Dpto. Comercial)

NOMBRE Y APELLIDOS

EMPRESA

DIRECCIÓN

FAX

TELÉFONO

POBLACIÓN

PROVINCIA

Comelta, s.a. INTERNET http://www.comelta.es

Ctra. de Fuencarral Km. 15,700 - Edificio Europa 1ºpl. - 1 • Tel.: (34 1) 657 27 50 • Fax: (34 1) 662 20 69 • E-mail: mad-informat@comelta.es • 28108 ALCOBENDAS (Madrid)

Avda. Parc Tecnològic, 4 • Tel.: (34 3) 582 19 91 • Fax: (34 3) 582 19 92 • E-mail: bcn-sti@comelta.es • 08290 CERDANYOLA DEL VALLÈS (Barcelona)

Rua do Entreposto Industrial nº3, sala E, Edificio Turia, Quinta Grande • Tel.: (351 1) 472 51 90 • Fax: (351 1) 472 51 99 • 2720 ALFRAGIDE (Portugal)



Autor: Jorge R. Regidor

Crear un control ActiveX

Los controles ActiveX nos permiten de manera fácil y eficiente reutilizar software ya creado y ampliar notablemente la funcionalidad de nuestras aplicaciones.

Cualquier aplicación que vayamos a desarrollar para entorno Windows con cualquier lenguaje o herramienta, llámese Visual Basic, Visual C++, Delphi,... utilizan componentes ya creados en forma de controles ActiveX. La introducción de este tipo de controles, no ha permitido crear aplicaciones con soporte de gráficas o informes, capacidad para el envío y recepción de mails, o cualquier otra funcionalidad más o menos avanzada con la simple introducción de un control que realiza el trabajo por nosotros.
El uso de controles ActiveX desde Visual C++ lo vimos muy brevemente en el artículo anterior, pero normalmente Visual C++ es utilizado para la creación de los controles ActiveX y es desde otros lenguajes más sencillos o visuales desde donde se utilizan. A día de hoy la creación de una aplicación Windows suele apoyarse en varias herramientas con un criterio más o menos común. Visual C++ se utiliza para crear componentes que requieren una velocidad y funcionalidad compleja y un lenguaje de tipo visual se utiliza a modo de pegamento para crear el interfaz de usuario y la funcionalidad de la aplicación.
En este artículo vamos a comenzar a describir el método a seguir para crear controles ActiveX, en un primer estadio desde la librería de clases MFC y el asistente AppWizard, y en un segundo estadio utilizando la librería ATL. Esta última alternativa, permite crear controles ActiveX de más reducido tamaño, y puede ser una alternativa muy a tener en cuenta para crear controles ActiveX sencillos para ser bajados desde Internet.

CREAR UN CONTROL ACTIVEX DESDE VISUAL C++

Anteriormente Visual C++ era la única herramienta, bueno mejor dicho, C/C++ eran los únicos lenguajes desde el que se podían crear controles ActiveX, pero actualmente se pueden crear controles ActiveX desde Visual Basic o Java, con mayor o menor grado de éxito.
Si enfocamos al entorno Visual C++, podemos crear controles ActiveX directamente en C/C++, utilizando las clases brindadas por las MFC al respecto o utilizar la librería ATL. Las dos últimas opciones tienen un Wizard desde AppWizard que nos simplifica de manera notable la creación de dichos controles. Vamos a entran sin más dilación a ver el método basado en las clases MFC.

MFC ACTIVEX CONTROL WIZARD

Para crear un control ActiveX desde las clases de MFC, debemos crear un nuevo proyecto y seleccionar la entrada MFC ActiveX Control Wizard, e indicar el nombre del proyecto a crear. En nuestro caso le vamos a denominar IconBarCtrl, ya que vamos a crear un control que nos brinde la funcionalidad para agregar y controlar iconos a la barra de tareas de Windows 95 y Windows NT.

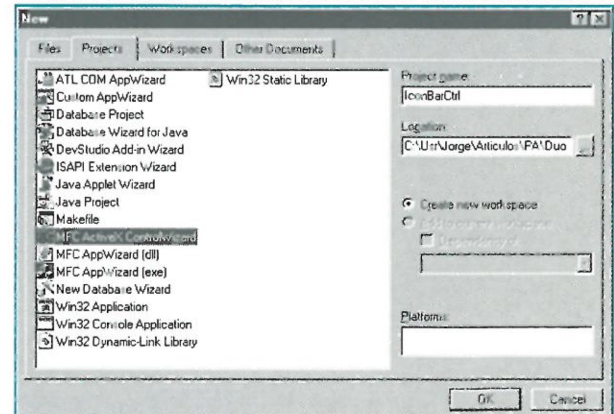
PASO 1 DE 2

Ya dentro del asistente para la creación de controles ActiveX, se muestran una serie de opciones a elegir que son las siguientes:
La pregunta que aparece en la parte superior, "How many controls would you like your project to have?", nos permite indicarle al asistente el número de controles incluidos en el mismo fichero OCX a crear. Es decir, por un mismo archivo OCX pueden existir diferentes controles, lo que nos permite crear conjuntos de controles con funcionalidad similar.
La siguiente pregunta, "Would you like the controls in this project to have runtime license?", nos permite crear controles ActiveX con licencia de ejecución para no permitir la libre ejecución del control. Al indicar Yes en esta opción, ControlWizard inserta una serie de funciones que nos permiten crear ficheros .LIC con la licencia de ejecución del control ActiveX.
La siguiente pregunta nos permite indicarle a ControlWizard si queremos que nos incluya o no comentarios sobre el código fuente C++ que nos va a generar.
La última opción de este diálogo, nos permite indicarle a ControlWizard si queremos crear ficheros de ayuda contextual asociado con el control AciteX.
En nuestro control ActiveX de ejemplo vamos a seleccionar un solo control, la opción para trabajar con licencias de ejecución, le vamos a indicar que nos cree los comentarios en el código fuente y por último le vamos a indicar que cree los ficheros de ayuda asociados.

PASO 2 DE 2

En este segundo paso se seleccionan opciones concretas para indicar el tipo control a crear. En la parte superior de la ventana aparece un ListBox con todos los controles a crear. Seleccionando cada entrada nos permite indicarle las diferentes opciones de cada control.
En el botón Edit Names, nos aparece un diálogo donde podemos indicar los diferentes nombres asociados al control ActiveX. En la casilla ShortName se puede indicar el nombre por defecto para los otros campos, si queremos cambiar todos debemos editar este campo.
En el diálogo podemos ver dos partes diferenciadas, una Control y la otra PropertyPage, donde se pueden editar los nombres asociados al control y al diálogo de edición de propiedades del control respectivamente.
En la sección Control se puede editar el nombre de la clase a crear en el código fuente en la casilla Class Name, el nombre del fichero de cabecera en Header File o el nombre del fichero fuente en Implementation File. En la casilla Type Name, se indica el nombre que se puede ver al insertar el control en un proyecto

FIGURA A. SELECCION DE APPWIZARD PARA CREAR UN CONTROL ACTIVEX.





desde la lista de controles ActiveX, y Type ID es el nombre con el que se va a registrar el control en el registro de Windows. En el caso del diálogo de edición de propiedades las casillas son exactamente las mismas que en el caso de Control.

Una vez indicado el nombre del control, los ficheros y las clases del ActiveX, se pueden configurar las diferentes opciones de cada uno. Las opciones a seleccionar son las siguientes:

- *Activates when visible*: Esta opción permite que el control ActiveX se active automáticamente cuando se visualiza en pantalla.
- *Invisible at runtime*: Permite crear controles ActiveX invisibles en tiempo de ejecución, pero visibles en tiempo de edición.
- *Avaible in "Insert Object" dialog*: Permite que el control aparezca en la lista de controles ActiveX a insertar en la aplicación contenedora.
- *Has an "About" box*: Crea un diálogo y un método About que muestra información sobre el control (Versión, autor, copyright, etc...)

En la ListBox inferior, *"Which Window subclass, if any, should this control subclass?"*, permite indicar de que control estándar de Windows vamos a derivar nuestro control. Podemos seleccionar none, en el caso que queramos partir de una ventana normal (CWnd) o los diferentes controles como BUTTON, EDIT, LISTBOX,...

OPCIONES AVANZADAS

Si pulsamos el botón Advanced en el cuadro de diálogo se abre una nueva ventana donde se pueden seleccionar algunas de las opciones avanzadas de los controles ActiveX, que son las siguientes:

- *Windowless activation*: Esta opción se utiliza cuando el control no necesita una ventana propia. En este caso se utilizan los servicios de ventana de la aplicación contenedora.
- *Unclipped device context*: Esta opción permite mayor velocidad de ejecución, pero hay que estar totalmente seguro de que no se va a pintar fuera del área de cliente del propio control.
- *Ficker-free activation*: Esta opción elimina las operaciones de dibujo al pasar de estado activado a desactivado. Esta opción sólo se debe habilitar si la imagen del control en modo activo y no activo es la misma.
- *Mouse pointer notifications when inactive*: Si se activa está opción, la aplicación contenedora delega el tratamiento de los mensajes asociados al ratón aún no estando activa.
- *Optimized drawing code*: Esta opción permite al control realizar operaciones avanzadas de dibujo si la aplicación contenedora lo permite.
- *Loads properties asynchronously*: Esta opción se debe habilitar cuando el control expone propiedades que son cargadas en background.

En nuestro control no vamos a activar ninguna opción avanzada, y tras terminar de ver todas las opciones de creación, pulsamos el botón Finish que permite crear el código fuente que va a definir el control ActiveX.

EL CODIGO FUENTE CREADO

Fruto de la ejecución del Wizard se crean las siguientes clases:

- CIconBarApp: Esta clase contiene las funciones de inicialización de la DLL, ya que el control no deja de ser una DLL, la función para registrar y dejar de registrar el control en Windows. Esta clase deriva de la clase ColeControlModule.
- CIconBarCtrl: Esta clase derivada de COleControl, contiene el núcleo del control y es la clase donde vamos a permitir crear los métodos, propiedades y eventos del control.
- CIconBarPropPage: Esta clase permite controlar las operaciones asociadas al control ActiveX del cuadro de diálogo de propiedades.

Además de las clases que definen el comportamiento del control, existen una serie de ficheros

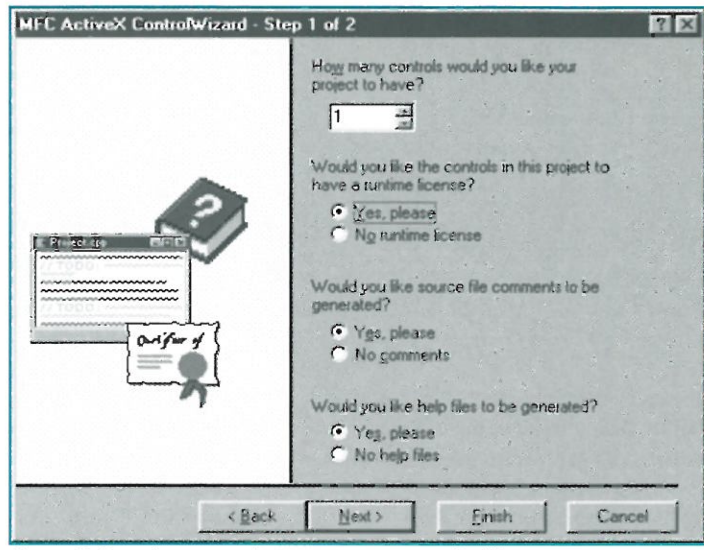


FIGURA B. PASO 1 EN EL DISEÑO DEL CONTROL DESDE APPWIZARD.

asociados que participan en el proyecto.

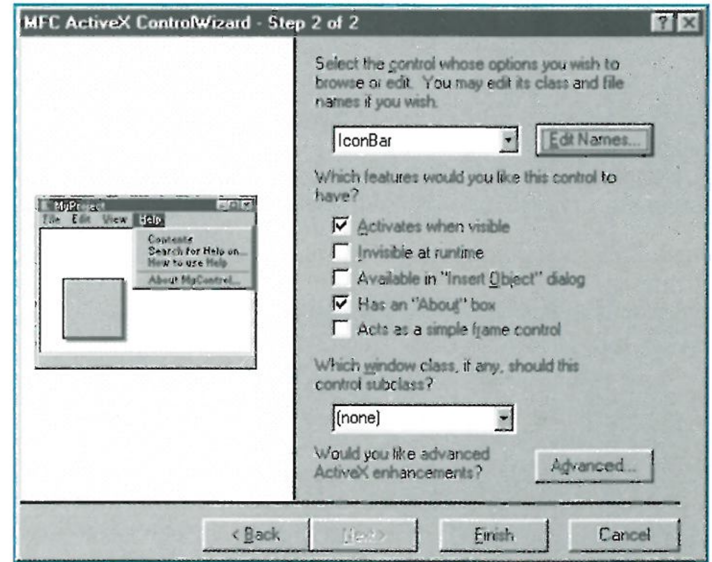
- IconBar.odl: Es el fichero fuente que define el interfaz del control ActiveX. Este fichero contiene los métodos, propiedades y eventos publicados por el control, y que serán compilados para generar un archivo de tipo TLB que formará parte de los recursos del archivo OCX.
- IconBar.def: Este es el fichero de definición, donde se definen las funciones exportadas por el control (DLL), así como el nombre del control al linkarse.
- IconBar.hpj: Este archivo que contiene las opciones de compilación de los archivos de ayuda.
- IconBar.rtf: Contiene el archivo de texto origen en formato RTF de la ayuda del control.
- IconBar.lic: Este archivo es la licencia de uso del control. Debe ser copiado al mismo directorio para poder trabajar con él en tiempo de edición.

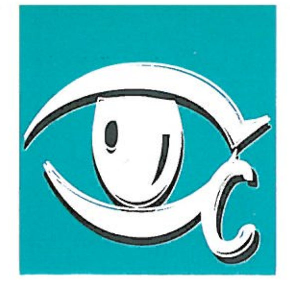
LA CLASE CICONBARAPP

La clase CIconBarApp deriva de la clase COleControlModule, clase que proporciona las funciones miembro para inicializar nuestro control ActiveX. Cada control creado con la librería MFC puede contener un solo objeto que derive de está clase.

El código fuente creado por AppWizard, nos crea dos

FIGURA C. SALIDA DEL ENLAZADOR EN MODO VERBOSE.





Curso Visual C

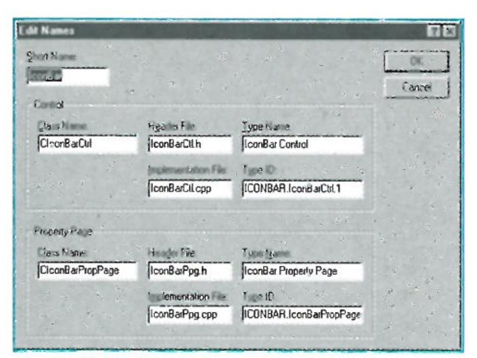


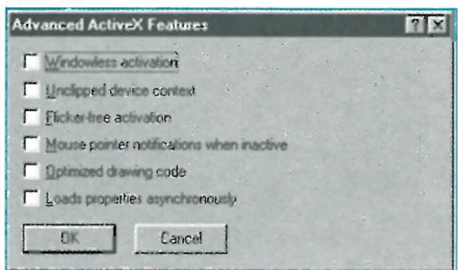
FIGURA D. VENTANA DE SELECCION DE NOMBRES.

funciones miembro que son InitInstance y ExitInstance, funciones miembro que son llamadas cada vez que una instancia del control es cargado o es descargado de memoria respectivamente.

Para crear un control ActiveX desde las clases de MFC, debemos crear un nuevo proyecto y seleccionar la entrada MFC ActiveX Control Wizard

Si nos fijamos en el código de InitInstance, lo primero que se realiza es una llamada a la función miembro InitInstance de la clase madre para permitir que dicha clase ejecute los procesos necesarios. En caso que dichos procesos se hayan creado correctamente, la función devuelve TRUE y en consecuencia con esto, podemos ya realizar nuestros propios procesos de inicialización en nuestro control.
En la función miembro ExitInstance el proceso es similar pero invertido, ya que primero somos nosotros los que debemos ejecutar nuestro código de finalización y luego llamar al método ExitInstance de la clase COleControlModule.
En cualquiera de los dos casos AppWizard nos orienta de donde insertar nuestro código con un comentario al respecto tal y como muestra la figura F.
Hay un detalle a tener en cuenta en el fichero de implementación de la clase CIconBarApp,

FIGURA E. OPCIONES AVANZADAS DE LOS CONTROLES ACTIVEX.



ya que el objeto asociado a dicha clase se debe crear en un ámbito global y sólo puede existir uno. En el comienzo del archivo IconBar.cpp se observa la siguiente línea:
CIconBarApp NEAR theApp;
La cual es responsable de crear dicho objeto, por tanto es un comentario a tener en cuenta, pero AppWizard ya nos a realizado también este trabajo.
En el mismo archivo (IconBar.cpp) existen también dos funciones ajenas a la propia clase y que implementan en interfaz que permite a Windows registrar nuestro control en el registro del sistema. Estas funciones son:

- DllRegisterServer: Función que permite registrar el control en el registro del sistema.
- DllUnRegisterServer: Función que permite eliminar el control del registro del sistema.

Ambas funciones están también creadas por AppWizard, y en principio nunca o casi nunca vamos a requerir cambiarlas, pero no está demás ver aunque sea de pasada lo que hacen ambas.
Si nos fijamos en la función DllRegisterServer, observamos las siguientes líneas:

```
STDAPI DllRegisterServer(void)
{
  AFX_MANAGE_STATE(_afxModuleAddrThis);

  if
(!AfxOleRegisterTypeLib(AfxGetInstanceHandle(),
_tlid))
    return ResultFromScode(SELFREG_E_TYPELIB);

  if
(!COleObjectFactoryEx::UpdateRegistryAll(TRUE))
    return ResultFromScode(SELFREG_E_CLASS);

  return NOERROR;
}
```

La llamada AFX_MANAGE_STATE protege la llamada a la función exportada cargando el módulo indicado, es decir el del propio control. La llamada a AfxOleRegisterTypeLib se encarga de registrar la librería de tipos asociada al control que define el interfaz exportado por el control a otras aplicaciones OLE. Los parámetros pasados a esta función son el handle a la instancia del control y el identificador universal de la librería de tipos.
La siguiente llamada a COleObjectFactoryEx:: UpdateRegistryAll con el parámetro a TRUE permite registrar el control en el sistema.

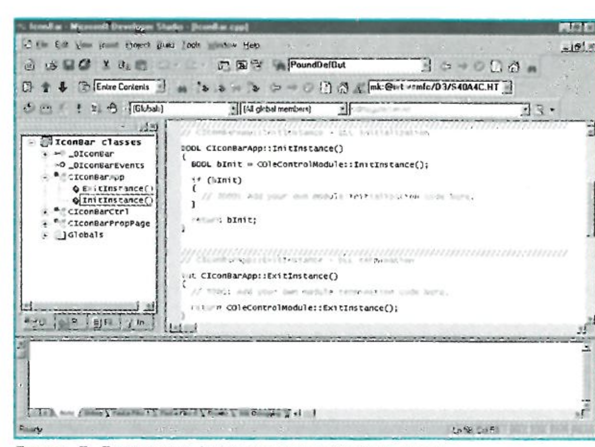


FIGURA F. FUNCIONES INITINSTANCE Y EXITINSTANCE.

En caso de existir algún tipo de error en las llamadas anteriores, el código de error se devuelve llamando a la función ResultFromScode.
Si Observamos del mismo modo la función DllUnregisterServer,

```
STDAPI DllUnregisterServer(void)
{
  AFX_MANAGE_STATE(_afxModuleAddrThis);

  if (!AfxOleUnregisterTypeLib(_tlid, _wVerMajor,
_wVerMinor))
    return ResultFromScode(SELFREG_E_TYPELIB);

  if (!COleObjectFactoryEx:: (FALSE))
  return ResultFromScode(SELFREG_E_CLASS);

  return NOERROR;
}
```

se observa como de manera paralela se llama a la función AfxOleUnregisterTypeLib para eliminar del registro la librería de tipos y a la función AfxOleUnregisterTypeLib, con el parámetro a FALSE para eliminar la entrada del control en el registro de Windows.

Siguiente artículo

Vamos a para en este punto el artículo, para continuar viendo en el siguiente artículo el código fuente generado por AppWizard. Una vez visto todo el código, vamos a ir definiendo diferentes métodos, propiedades y eventos del control, en definitiva, construyendo un control ActiveX real.
Además, y aprovechando la creación del control, vamos a generar un control que nos permita controlar los iconos de la barra de Windows 95/NT. Como iremos viendo, es una funcionalidad muy sencilla pero que puede dar un toque de distinción a las aplicaciones siguientes que desarrollemos.

Capítulo 2º
Autores: Mª Jesús Recio y José M. Jiménez



Configuración del sistema

La complejidad de los sistemas operativos actuales hace que la fase de instalación de éstos haya perdido la importancia que antiguamente tenía. En la actualidad, la fase más crítica y de la que depende el correcto funcionamiento de la máquina es la de configuración del sistema.

Una vez vista la instalación del sistema, lo más conveniente es empezar a estudiar, punto por punto, todos los elementos necesarios para la configuración y optimización del mismo.
Bajo versiones de Windows para MS-DOS, las acciones más comunes como el inicio de sistema, la conexión a la red o la ejecución de aplicaciones conllevan el uso y la sincronización de varios ficheros de configuración, tales como AUTOEXEC.BAT, CONFIG.SYS, WIN.INI, SYSTEM.INI, PROTOCOL.INI, etc. Sin embargo, en Windows NT la información de configuración referente al sistema se almacena y se consulta en lo que se llama el Registro.
Para hacerse una idea de qué contiene el Registro, piense que cualquier cambio que se realice a través del Panel de Control (ya sea agregar o quitar componentes hardware o software, configurar la pantalla o el inicio y apagado del sistema, el manejo de la memoria virtual, etc.) o a través del Editor de directivas del sistema (herramienta para acceder al Registro de una forma más amplia que el Panel de Control), queda reflejado en la base de datos que es el Registro.
Como se puede deducir de lo expuesto hasta ahora, se pueden modificar los valores contenidos en el Registro a través del Panel de Control o del Editor de directivas del sistema, pero siempre de una manera controlada.
Además de estos dos caminos existe otro: el propio Editor del Registro que, como su propio nombre indica, permite editar el registro, dando la posibilidad de una manipulación completa del mismo; piense que al editar la base de datos puede realizar cualquier operación sobre ella, pudiendo, por tanto, introducir valores incorrectos o borrar datos con el consiguiente efecto sobre el sistema. Teniendo esto en cuenta, se puede decir que, de estas tres herramientas, el Panel de Control es la más segura pero tiene como contraposición que es la que menos componentes del Registro deja configurar, y el Editor del Registro la que más componentes abarca pero la más arriesgada.
Aunque pudiera parecer que la mejor forma de informar al sistema de un cambio en su configuración sería la edición del Registro, es la más insegura, ya que el Editor de Registro no reconoce un error de sintaxis que se pudiera cometer al introducir un valor erróneo para, por ejemplo, una variable de entorno o la resolución de nuestra pantalla.
Por tanto, siempre que no sea totalmente necesario, se deberá evitar la edición del Registro y se utilizarán o el Panel de Control o el Editor de Directivas del sistema, que saben cómo almacenar valores en el Registro.

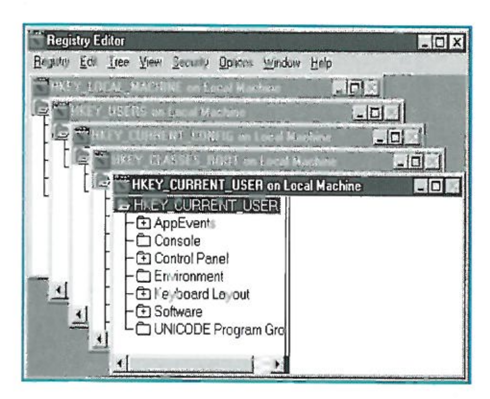
EL REGISTRO

Tal vez se esté preguntando qué es eso del registro que parece tener tanta importancia en este sistema. Pues bien, ha llegado el momento de averiguarlo.
El registro es una base de datos que almacena información del sistema de una forma jerárquica. Está dividido en un conjunto de cinco subárboles, que se visualizan de una forma similar a la manera en que el Explorador de Windows muestra los directorios y sus contenidos, y que contienen información referente a la máquina y a los usuarios. En la figura 1 se puede observar la citada estructura jerárquica del Registro y en la tabla 1 se citan los diferentes subárboles del Registro, así como una breve descripción de cada uno de ellos.
Cada subárbol está, a su vez, dividido en subclaves; en la tabla 2 aparecen reflejadas algunas de estas subclaves para cada uno de los cinco subárboles mencionados.

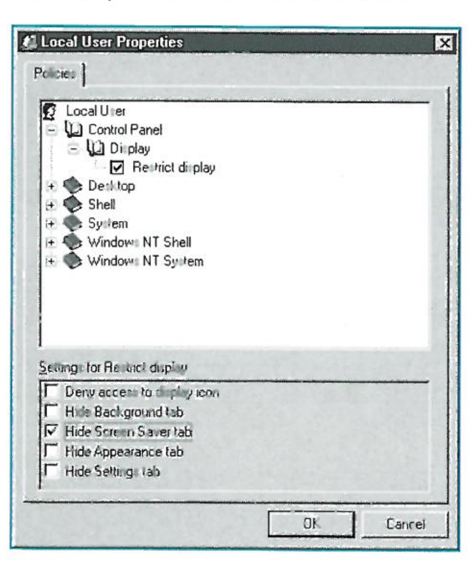
EL EDITOR DEL REGISTRO

Es hora de conocer cómo se accede al registro para consultarlo o para cambiar su configuración. Para ello se debe ejecutar el mencionado Editor del Registro. Este programa, REGEDT32.EXE, se encuentra en el directorio \WINNT\SYSTEM32, pero no forma parte de ninguno de los programas que aparecen en el menú de Inicio; por esta razón, es recomendable crear un acceso directo que facilite la edición del Registro.
Los administradores del sistema (el propio administrador y los usuarios pertenecientes a los grupos *Administrators* y *System*) tienen acceso total al Registro, mientras que el resto de los usuarios, por defecto, tienen un acceso restringido a sólo lectura. Como ya se ha mencionado, con el Editor del Registro se cubre la totalidad del Registro pero se corre el riesgo de introducir algún dato que haga funcionar al sistema de manera incorrecta. Para evitar en lo posible este tipo de problemas existen las posibilidades de tener un acceso al Registro en modo sólo lectura, de pedir confirmación si se borra algún dato o de salvar o no los cambios al salir. Este tipo de precauciones está disponible en el menú Opciones del Editor del Registro, y deberían ser configuradas nada más ejecutar REGEDT32.EXE.
El Editor del Registro incorpora facilidades que proporcionan un sencillo manejo del Registro en sí como, por ejemplo, las referentes a la manipulación de las subclaves: búsqueda, guardado, restauración. La tabla 3 muestra la utilidad de estas herramientas.
A veces es necesario que usuarios que no pertenecen al grupo Administradores ni al grupo System puedan acceder al Registro de una forma menos restringida que la de por defecto, es decir, que puedan hacer con el Registro algo más que verlo solamente. La forma de modificar este tipo de acceso, que es muy similar a la forma de añadir usuarios al sistema, a grupos o a la asignación de permisos, tiene que seguir estos pasos:
Lo primero que se ha de hacer es situarse en el subárbol o subclave sobre la que se quiera realizar las modificaciones, es decir, cambiar los





permisos o añadir nuevos tipos de acceso. Al seleccionar la opción Permisos del menú Seguridad aparece la ventana Permisos de la clave del Registro que muestra los usuarios y grupos que tienen acceso al Registro y el tipo de acceso para cada uno de ellos. Por defecto, y si no se modifican los valores después de la instalación, en la lista de usuarios aparecen el administrador del sistema y los grupos Administradores y System con un control total sobre el Registro. Esta situación se puede cambiar con sólo seleccionar uno de los usuarios o grupos, y modificar el valor de la lista desplegable Tipo de acceso. Si queremos que otros usuarios o grupos tengan acceso al Registro, se debe pulsar el botón Añadir que nos mostrará la ventana Añadir Usuarios y Grupos tal y como puede observarse en la figura 2. En esta ventana se pueden seleccionar grupos completos o usuarios en concreto. Para esto último hay que pulsar el botón Mostrar usuarios. Una vez que se tiene seleccionado el usuario o grupo en cuestión, se añade a la lista de nombres y se elige el tipo de acceso: sólo lectura, control total o tipo de acceso especial. En este último tipo se pueden seleccionar para un usuario o un grupo las opciones de establecer un valor, crear una subclave o un enlace, borrar, hacer una petición de un valor en concreto,...



EL EDITOR DE DIRECTIVAS DEL SISTEMA

El Editor de directivas es el siguiente paso para acceder al Registro, si se quiere configurar algo a lo que no se puede tener acceso desde el Panel de Control. Se debe tener cuidado con lo que se modifica en el Editor de directivas ya que, aunque este programa sabe cómo almacenar los datos en el Registro, puede que la opción elegida por el usuario traiga como consecuencia que el equipo no sea capaz de iniciarse.

El Editor de directivas del sistema sólo forma parte de Windows NT Server y no está disponible en Windows NT Workstation. Se encuentra dentro de las Herramientas administrativas de los Programas del menú de Inicio (*System Policy Editor*) y tiene dos modos de funcionamiento: modo registro y modo archivo de directivas.

En el modo registro, el administrador puede editar el Registro de la máquina local o el de una máquina remota y puede cambiar los valores referentes al equipo o al usuario que en ese momento haya iniciado una sesión en la máquina. Si se accede a las propiedades de alguno de ellos se podrá observar algo parecido a lo que se muestra en la figura 3. En dicha figura se ve cómo restringir al usuario local el acceso a las propiedades de la pantalla, de tal manera que no podrá elegir, entre otras cosas, su salvapantallas porque esa ficha no aparece en el cuadro de diálogo de la configuración de la pantalla. Dentro de estas propiedades también se puede restringir otro tipo de aspectos referentes, por ejemplo, al escritorio (se puede definir el fondo que éste tendrá especificando un fichero que lo contenga), o a la Shell (es posible eliminar el comando ejecutar del menú de Inicio, la barra de tareas de la opción Configuración del mismo menú, la utilidad de Búsqueda, o deshabilitar la posibilidad de reiniciar la máquina). Todas estas opciones cobran más significado sabiendo que se puede modificar el Registro de otra máquina mediante el Editor de directivas. Esto se hace desde el menú Archivo del editor y con la opción Conectar, que proporciona una manera de controlar, al administrador, lo que pueden hacer los usuarios pertenecientes a su dominio.

En el modo archivo de directivas, se crean una serie de directivas que se almacenan en el fichero *NTConfig.pol*, ubicado en el directorio *\Winnt\System32\Repl\Import\Scripts* y que tomarán efecto cuando el usuario para el que se hayan definido, inicie una sesión. La forma de crear este fichero es abriendo una nueva directiva desde el menú Archivo, añadiendo usuarios o grupos a los que se quiere que afecten dichas directivas, y definiendo sus propiedades. Dichas directivas consisten en una serie de reglas que definen lo que un usuario puede hacer en un equipo. Estas opciones son las mismas a las que se puede acceder desde el modo registro visto anteriormente. En la figura 4 se muestra la pantalla desde la que se pueden añadir usuarios o grupos para, posteriormente, modificar sus propiedades.

Esta pantalla acabará resultando familiar ya que, siempre que se trate con grupos y usuarios a los que se les quiere dar unos atributos o unos permisos sobre una determinada parte del sistema, el administrador se encontrará con una ventana similar.

Las directivas definidas toman efecto siguiendo un determinado proceso: al iniciarse una sesión por parte de un usuario el sistema busca el fichero *NTConfig.pol.* Si existe y se han definido directivas para el usuario, éstas se combinan con la parte de usuario actual del Registro. Si en vez de para usuario, se han definido para un grupo al que pertenece el usuario, éstas directivas se combinan igualmente con la parte de usuario actual del Registro. Si el usuario no se encuentra en ninguno de estos casos se tomarán los valores del usuario predeterminado. Si se han definido directivas para el equipo local tomarán efecto ahora; en caso contrario, se utilizarán las del equipo predeterminado.

DOMINIOS Y GRUPOS DE TRABAJO

Recuerde que un dominio consiste en una agrupación de máquinas y usuarios. A veces, se hace necesaria la división de los recursos de la red en departamentos separados, de forma que un usuario tenga acceso a ciertos recursos, pero no a todos. Cada uno de esos departamentos es un dominio. Cuando un usuario se conecta a la red, debe seleccionar el dominio al que quiere entrar de todos a los que le está permitido acceder. Al ser autentificado en un dominio, el usuario tiene disponibles todos los recursos dados de alta en dicho dominio, sin tener que autentificarse en cada uno de los servidores que formen parte de dicho dominio, siendo, inicialmente, transparente para él los recursos que pertenecen a otros dominios. La gestión de un dominio se realiza de forma centralizada, ya que la información de todos los integrantes de éste está en una base de datos almacenada en el PDC.

Por tanto, se puede concluir que en una misma red pueden existir varios dominios, cada uno de ellos integrado por una serie de recursos y usuarios, y que cada vez que se quiera entrar a un dominio diferente es necesario (si no existe una relación de confianza establecida entre dominios) identificarse.

Conceptualmente un grupo de trabajo es lo mismo que un dominio. La diferencia entre ellos está en la administración de los mismos. En un grupo de trabajo la gestión de los recursos y de las cuentas de usuario se lleva a cabo en cada equipo que forma parte del



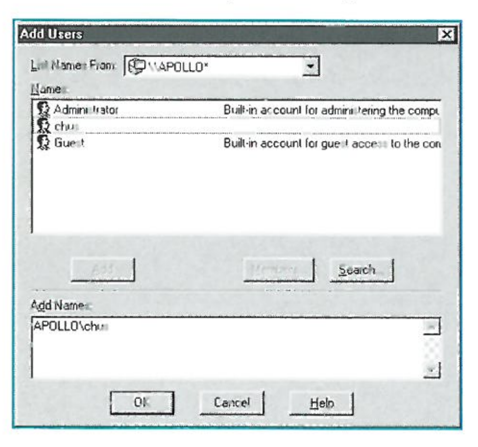
TABLA 1. SUBARBOLES DEL REGISTRO

HKEY_LOCAL_MACHINE	Contiene todos los datos sobre la configuración de la máquina. Estos datos son fijos, independientemente del usuario, y determinan, entre otras cosas, qué controladores de dispositivos y qué servicios durante el inicio.
HKEY_CLASSES_ROOT	Contiene datos de configuración del software instalado en la máquina. Sus valores coinciden con los de la subclave referente a software del subárbol HKEY_LOCAL_MACHINE.
HKEY_CURRENT_USER	Contiene dados sobre el usuario que ha iniciado la sesión. Sus valores se modifican cada vez que se inicia el sistema con los datos introducidos por el usuario.
HKEY_USERS	Está dividido en dos subclaves: una con los valores por defecto que se utilizan hasta que se identifica el usuario, y otra variable que almacena el identificador del usuario que inicia la sesión, denominado identificador de seguridad (SID).
HKEY_CURRENT_CONFIG	Contiene información sobre el perfil de hardware activo.

grupo, es decir, de forma distribuida, en contraposición a la centralización de la base de datos en el caso de los dominios.

CONTROLADOR PRINCIPAL DE DOMINIO (PDC)

Como ya se dijo, el controlador principal de dominio mantiene la base de datos de usuarios del dominio y debe ser único en él, para lo cual se debió configurar la máquina elegida como Controlador Principal de Dominio (PDC) en el proceso de instalación de Windows NT Server.
Los cambios que se realicen en la herramienta administrativa del manejador de usuarios del dominio quedarán reflejados en la base de datos del PDC. Estos cambios (creación de usuarios o grupos, pertenencia de usuarios a determinados grupos, concesión de permisos a o grupos,...), se llevan a cabo seleccionando el dominio sobre el que se quiere actuar que, en el caso de tener establecida una relación de confianza entre ambos, bastaría con seleccionar dicho dominio, pero en el supuesto de no existir dicha relación, será necesario identificarse en el nuevo dominio. Fíjese que se selecciona el dominio y no la máquina que realiza las funciones de PDC, que es la que tiene la base de datos.
Más adelante se verá cómo tener varios dominios en la red con sus correspondientes controladores a través de las relaciones de confianza.
Es necesario recordar que una vez que se ha



configurado una máquina como controlador principal de dominio (opción que se elige al instalar Windows NT Server) no puede pasar a ejecutar tareas como la de controlador de reserva o la de servidor autónomo sin tener que volver a instalar el sistema en ella.
Para aclarar esto se va a poner un ejemplo práctico. Suponga el lector que se tiene instalado un PDC, como único elemento del dominio. Ahora, si se apaga el PDC y se instala una nueva máquina como controlador principal, indicándolo durante la instalación, no se producirá ningún error ya que el dominio no tenía ningún PDC activo. Posteriormente, se quiere que la máquina que dejó de prestar servicio vuelva a formar parte del dominio para lo cual se enciende de nuevo el antiguo controlador principal, produciéndose un error al detectarse la existencia de otro PDC en el dominio. La única solución que le queda al administrador del dominio para poder aprovechar la máquina que se configuró como PDC originalmente en el dominio actual es volver a instalar Windows NT Server en ella y configurarla como BDC o como servidor autónomo.
El problema estaría solucionado si se pudiera descender de categoría al PDC original para que ejerciera funciones de BDC. Esto no es posible porque cuando se instala un controlador principal también se le asigna un identificador de seguridad, SID, que forma parte del propio identificador de los BDC's y de las cuentas de usuario. Por eso, cuando se intenta que el controlador principal inicial vuelva a forma parte del dominio, su identificador de seguridad no es reconocido por el controlador actual.
Para que este problema no llegue a producirse se debe tener la precaución de que, una vez que se tiene un PDC para el dominio, no volver a configurar ninguna máquina como PDC aunque el original esté fuera de uso. Los controladores de reserva tienen la posibilidad de promocionarse a controladores principales, por lo que siempre habrá una máquina que se esté comportando como PDC.

CONTROLADOR DE RESERVA DE DOMINIO (BDC)

Después de conocer mejor el funcionamiento de los PDC's es necesario saber cuál es la utilidad de los controladores de reserva o *backup*.
Ya se ha comentado que la información de seguridad referente a las cuentas de los usuarios de un dominio es mantenida en una base de datos centralizada gestionada por un controlador principal de dominio. Esta información se encuentra duplicada en los diferentes controladores de reserva que se hayan asignado para el dominio. Las máquinas que se vayan a comportar como controladores de reserva (o BDC's) deben tener instaladas Windows NT Server y ser configuradas como controladores BDC's en el proceso de instalación.
Los controladores de reserva, al igual que los principales, tienen la habilidad de gestionar las peticiones de inicio de sesión en el dominio por parte de los usuarios que tengan cuentas definidas en el mismo. Cuando un usuario introduce su nombre y su clave puede ser autentificado tanto por un controlador principal como por uno de reserva.
La presencia de los controladores de reserva del dominio se justifica por dos razones. La primera, y más importante, es la seguridad que proporcionan al dominio. Si un controlador principal deja de funcionar por cualquier razón, un controlador de reserva puede promocionarse a PDC y hacer así que el dominio siga funcionando.
La segunda de las razones por la que los controladores de reserva son necesarios es que permiten aligerar la carga de trabajo de los controladores principales, en cuanto a la gestión de inicio de sesión en el dominio. Los controladores de reserva también pueden validar a los usuarios que quieran formar parte del dominio en un momento determinado introduciendo su nombre y su clave.

SERVIDORES AUTONOMOS

Al igual que en los casos de los controladores principales y de reserva del dominio, los servidores



TABLA 2. DIFERENTES SUBCLAVES DE CADA UNO DE LOS SUBARBOLES

HKEY_LOCAL_MACHINE\Hardware	Contiene parámetros volátiles que se obtienen cada vez que se inicia el equipo.
HKEY_LOCAL_MACHINE\Software	Contiene información sobre el software del equipo local.
HKEY_CLASSES_ROOT	El propósito de este subárbol es la compatibilidad con el registro de Windows 3.1. Contiene la misma información que *HKEY_LOCAL_-MACHINE\Software\Classes*
HKEY_CURRENT_USER\Printers	Describe las impresoras instaladas para el usuario actual. Esta información es cargada cuando se instala una impresora desde la ventana Impresoras del menú Configuración-Inicio.

autónomos son máquinas que están ejecutando Windows NT Server y que en el proceso de instalación se decidió, para estos casos, que se comportaran como servidores autónomos.
Este tipo de servidores no es necesario para el correcto funcionamiento de un dominio. Si un servidor autónomo pertenece a un dominio, puede disponer de todas las facilidades en cuanto a creación de usuarios o asignación de permisos que la base de datos centralizada de dicho dominio proporciona. Los servidores autónomos que participan en un dominio también disponen de las cuentas de usuario de los dominios en los que el dominio al que pertenece confíe, pero no tienen una copia de la base de datos del controlador principal, ni procesan los intentos de inicio de sesión de los usuarios en el dominio. Esta última característica puede ser una buena razón para tener servidores autónomos formando parte de un dominio; al no tener que perder tiempo validando cuentas de usuarios, el servidor puede dedicarse a tareas que necesiten un tiempo de respuesta muy corto.
La otra posibilidad es tener un servidor autónomo que ejecute Windows NT Server y que no pertenezca a ningún dominio. En este caso, el servidor mantendrá su propia base de datos de usuarios y debe ocuparse de validar las peticiones de inicio de sesión por parte de los usuarios creados por el mismo. Asimismo, no tiene acceso a las posibles cuentas de usuarios creadas en cualquier dominio. Esta forma de trabajar es muy semejante a la de los ordenadores con Windows NT Workstation instalado.

MAQUINAS CON WINDOWS NT WORKSTATION

Para el caso de máquinas con Windows NT Workstation instalado existe la posibilidad de integrarlos en un dominio o en un grupo de trabajo. Si se decide que el equipo forme parte de un dominio, se podrá iniciar una sesión con una cuenta de usuario que se encuentre en la base de datos mantenida por los controladores principal y de reserva de dicho dominio.

DOMINIOS Y RELACIONES DE CONFIANZA

Para explicar en qué consisten las relaciones de confianza se debe pensar en un dominio como un grupo de servidores que tiene instalados Windows NT Server y que forman un único sistema junto con las posibles máquinas clientes como Windows NT Workstation, Windows 95, Windows 3.11,...
Todos los servidores Windows NT Server del dominio comparten el mismo conjunto de cuentas de usuario, por lo que únicamente es necesario guardar la información de una cuenta para que tanto el controlador primario como los secundarios reconozcan dicha cuenta.
Las relaciones de confianza permiten la compartición de las bases de datos de usuarios entre varios dominios de la red; es decir, establecen un vínculo o relación por la que una cuenta de un dominio será reconocida por los servidores de los dominios que confíen en él. Mediante estos vínculos, un usuario tendrá una única cuenta en un dominio, pero podrá acceder a cualquier servidor de la red formada por los dominios que constituyen la relación.
Gracias a las relaciones de confianza, al crear una cuenta de usuario en un dominio, ésta queda habilitada en todos los servidores de dominio de la red de confianza.
Al establecerse una relación de confianza las cuentas de usuario del dominio en el que se confía son reconocidas por el dominio que da esa confianza, por lo que las cuentas del dominio en el que se confía pueden iniciar sesiones, pertenecer a grupos o recibir permisos por parte del dominio que confía. Es decir, se comportan como si pertenecieran al dominio que ha establecido la confianza. Es importante distinguir entre el dominio que confía (el que establece la confianza) y el dominio en el que se confía, ya que tal y como se han presentado las relaciones de confianza hasta ahora, éstas son unidireccionales. Esto quiere decir que las cuentas del dominio que confía no son reconocidas por los servidores del dominio en el que éste confía. Para que esto sea así, se debe establecer otra relación de confianza en sentido contrario, por lo que la relación pasará a ser bidireccional (un par de relaciones unidireccionales).
También es necesario aclarar que si un dominio confía en otro, y éste último confía en un tercero, el primero no confía automáticamente en el tercero. Es decir, las relaciones se establecen entre parejas de dominios o lo que es lo mismo, la confianza entre dominios no es una operación transitiva.

TABLA 3. FACILIDADES DEL EDITOR DEL REGISTRO

Buscar clave	Busca claves en el árbol seleccionado, de forma descendiente o ascendente. No busca valores.
Guardar clave	Guarda claves y subclaves en formato binario.
Restaurar	Carga los datos del fichero que contiene las claves y subclaves almacenadas por la opción Guardar clave.
Guardar subárbol como	Guarda claves y subclaves en un fichero de texto, lo que permite localizar valores.
Seleccionar equipo	Permite el acceso al registro de una máquina remota, para lo cual se debe pertenecer al grupo Administradores en el caso de Windows NT Server. Para Workstation, el acceso está permitido para cualquier cuenta de usuario válida.

Próximo número

En el próximo número se profundizará en los tipos de relaciones de confianza existentex, y los conceptos de grupos de trabajo local y global. También se describira el proceso de inicio de una sesión.

FORO LINUX

Nivel: Avanzado



Autor: Francisco Moya



Control de versiones con CVS

Llevar un control riguroso de las versiones de todos los ficheros de un proyecto es imprescindible en el desarrollo de programas. Pero también resulta útil en actividades cotidianas, como la administración de sistemas o la edición de documentos. En este artículo se describirá el manejo de CVS, una de las herramientas libres más completas para el control de versiones.

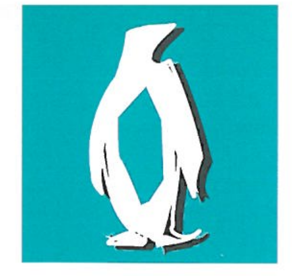
Todos los usuarios de ordenadores nos hemos enfrentado alguna vez con los quebraderos de cabeza que supone tener múltiples versiones de un documento o de un programa (o lo que es peor, no disponer de las versiones antiguas por no haberlas guardado). Al principio, se suele optar por poner un sufijo como 'intro.nuevo' o 'intro.último', pero pronto ese sufijo degenera en cosas como 'intro.ultimísimo'. La solución evidente es utilizar números de versiones ('intro-1.0', 'intro-1.1'). Hasta cierto punto, este esquema funciona razonablemente bien con proyectos diminutos pero existe un inconveniente que se hace cada vez más importante con proyectos más complejos: guardar todas las versiones de un fichero supone multiplicar, innecesariamente, el espacio ocupado en el disco cuando las diferencias entre versiones pueden ser realmente pequeñas. Parece mucho más sensato guardar las diferencias entre versiones. En cualquier variedad de GNU/Linux se pueden obtener las diferencias con el comando *'diff'* y se pueden recomponer las versiones completas con el comando *'patch'*. Ahora, sería más eficiente el almacenamiento pero cada vez más complicado guardar las versiones, y aún más engorroso recomponer cualquier versión a partir del conjunto de diferencias.
Hasta el momento hemos hablado de proyectos muy sencillos, de un solo fichero. Pero hasta los programas más simples suelen estar compuestos de varios ficheros; incluso en el caso de los documentos se pueden encontrar divisiones (figuras, capítulos, etc.) y esto complica bastante el manejo de versiones. Si sólo se modifica uno de los ficheros del programa ¿hay que cambiar la versión del programa, o hay que tener una versión por cada fichero? Parece evidente que habría que mantener una versión por fichero, pero veamos algún caso real: por ejemplo, el sistema de compilación de GNU (gcc) consta de, aproximadamente, 5000 archivos (incluyendo *libstdc++*). Para delimitar exactamente una versión habría que decir las 5000 versiones de los 5000 archivos. Entonces ¿qué significa exactamente gcc-2.8.0? En realidad, este número de versión es una simple etiqueta; se podría haber llamado gcc-98. Es una etiqueta que corresponde a un determinado estado del código, es decir, corresponde a un conjunto de unas 5000 versiones de ficheros.
Otro ejemplo: los proyectos reales no evolucionan de manera lineal, sino que se producen ramificaciones en el desarrollo. Así, mientras se trabaja en corregir errores en gcc-2.8.0, también se trabaja en añadir nueva funcionalidad y reorganizar el código de gcc. Los cambios más profundos posiblemente no se integrarán en la versión distribuida hasta que esté bien estable, cosa que puede no ocurrir nunca. Sólo hemos empezado a acercarnos a la realidad y ya empieza a ser totalmente incontrolable la gestión de versiones de forma manual.

HERRAMIENTAS DE CONTROL DE VERSIONES

Afortunadamente, existe un buen número de herramientas para ayudar en el control de versiones. Las tradicionales en el mundo Unix son SCCS (la primera, estándar pero muy ineficiente) y RCS (la versión de GNU, mucho más eficiente pero incompatible con SCCS). Ambas siguen un modelo denominado *'checkin-checkout'*. En este modelo existe un repositorio (normalmente un directorio), en el que se guardan todas las versiones de cada archivo de forma eficiente (una versión completa y las diferencias) y se proporcionan dos operaciones básicas:

- *'check-in'* o validación de una nueva versión, lo que implica la introducción de los cambios realizados en el repositorio.
- *'check-out'* o extracción de una versión concreta del repositorio.

Este modelo resulta muy adecuado para uso personal, pero no se adapta bien a grupos de trabajo. Si dos personas están realizando a la vez modificaciones sobre un archivo habrá todo tipo de problemas cuando se realice la validación (*'check-in'*) puesto que los cambios de uno pueden colisionar con los del otro. En la práctica, esto se resuelve imponiendo un acceso exclusivo al repositorio, es decir, mientras algún usuario haya realizado un *'check-out'* no se permite otro *'check-out'* hasta que ese usuario ejecute un *'check-in'*. Esto dificulta el trabajo en equipo sobre el mismo código.
En el mundo del software libre ha tenido mucho más éxito una evolución del modelo primitivo denominada *'copy-edit-merge'*. En este caso, se permite hacer múltiples operaciones de *'check-out'* y, posteriormente, la validación de los cambios tratará de integrar los cambios de múltiples usuarios de manera inteligente, resolviendo los conflictos de manera interactiva. La operación de *'check-out'* equivale a realizar una copia local de una determinada versión con todos los datos necesarios. La operación de *'check-in'* se sustituye por una operación de mezclado (*'merge'*) que resuelve la mayoría de los conflictos de forma automática.
Este cambio tan sencillo en el modelo permite todo un nuevo abanico de posibilidades, como la operación en modo cliente-servidor entre ordenadores distintos, lo que facilita enormemente el trabajo en equipos geográficamente dispersos. Cada ordenador cliente solicita una copia de la versión deseada y, cuando se han realizado cambios significativos, puede volver a conectarse con el repositorio central para refrescar la información del repositorio. Es perfectamente viable para ordenadores que no están habitualmente conectados a Internet.
Hay varias herramientas libres que siguen este modelo (CVS, PRCS, Aegis, Sup, etc.) pero, con mucho, la más popular de todas es CVS.




```
xterm
fmoya@segura:reactor$ cvs -d ~/repo init
fmoya@segura:reactor$ ls
Makefile            reactor.h          sock_handler.h     test_sock_clnt.c
reactor.c           sock_handler.c     test_reactor.c     test_sock_srv.c
fmoya@segura:reactor$ cvs -d ~/repo import reactor allabs version_inicial
N reactor/Makefile
N reactor/reactor.c
N reactor/reactor.h
N reactor/test_reactor.c
N reactor/sock_handler.c
N reactor/sock_handler.h
N reactor/test_sock_srv.c
N reactor/test_sock_clnt.c

No conflicts created by this import

You have mail in /home/fmoya/Mailbox
fmoya@segura:reactor$
```

FIGURA 1.

PRIMEROS PASOS CON CVS

Aunque CVS está diseñado para soportar el desarrollo concurrente de grandes sistemas software, eso no significa que no esté indicado para tareas más simples. Desde el punto de vista del usuario tiene un único ejecutable *'cvs'* que sirve para realizar todas las operaciones. El interfaz estándar de CVS utiliza comandos en línea de órdenes, aunque existen varios interfaces gráficos o pseudo gráficos que veremos en entregas posteriores.

Un poco de historia

CVS (Concurrent Versions System) nace en 1989 de la mano de Brian Berliner a partir de unos scripts publicados en Usenet News. Desde entonces, se han sucedido multitud de cambios realizados por un buen número de empresas especialmente Cygnus Support (www.cygnus.com) y Cyclic Software (www.cyclic.com). En la actualidad, la práctica totalidad de los cambios recientes en CVS son obra de Cyclic.

CVS inicialmente se apoyaba en otras herramientas como RCS (Revision Control System) o 'diff' que realizaban gran parte del trabajo, pero por razones de eficiencia estas dependencias se están eliminando progresivamente.

CVS es parte oficial del sistema GNU y, además, es utilizado por la mayoría de los proyectos relacionados con el software libre (gcc, emacs, guile, gtk, gimp, gnome, linux, por nombrar sólo unos pocos).

En la actualidad hay dos compañías de software libre que soportan y mantienen CVS: la sueca Signum Support (http://www.signum.se/) y la norteamericana Cyclic Software (http://www.cyclic.com/). Ambas se dedican a personalizar CVS para adaptarlo a los ciclos de vida de sus clientes automatizando la mayor parte de la burocracia (generación de informes, aprobaciones, etc.)

El primer paso, antes de poder utilizar CVS, es preparar un repositorio. Un repositorio es un directorio en el que se almacenan las versiones de los ficheros y los metadatos (datos de control para CVS). Esta operación se realiza con el comando *'cvs init'*. Por ejemplo:
mkdir ~/repo
cvs -d ~/repo init
Se ha utilizado la opcion *'-d'* de cvs para indicar dónde se encuentra el repositorio que se desea utilizar. También se puede poner este directorio en la variable de entorno CVSROOT y, en ese caso, no sería necesario especificarlo. En general, es recomendable usar *'-d'* si se utilizan varios repositorios porque es menos propenso a error.
Ahora ya se puede utilizar el repositorio. El proyecto más sencillo con el que puede trabajar CVS consta de un módulo. Un módulo simple para CVS es un directorio que tiene código más o menos independiente del resto. Por ejemplo, supongamos que estamos trabajando en el módulo *'reactor'* de otro proyecto mayor. El código inicial está en el directorio *~/devel/reactor* y se mete en el repositorio con el comando *'cvs import'*:
cd ~/devel/reactor
cvs -d ~/repo import reactor allabs version_inicial
Los argumentos de *'cvs import'* indican, por este orden, el nombre que queremos poner al módulo, el nombre de la organización y una etiqueta que identifica esa versión (*release*) para todos los ficheros. En nuestro ejemplo la versión se llama *'version_inicial'*. Justo antes de meter la información en el repositorio CVS invocará un editor para introducir un texto explicativo. El editor invocado será el definido en la variable de entorno VISUAL (por defecto, vi).
Ahora ya está almacenado todo bajo el control de CVS. Se puede borrar el directorio *~/devel/reactor* puesto que ya está bajo el control de CVS. Para volver a trabajar sobre ese programa será preciso pedirle una copia fresca a CVS utilizando el comando *'cvs checkout'*. No sirve utilizar la versión antigua de *~/devel/reactor* porque esa copia no está controlada por CVS:
rm -rf ~/devel/reactor
cd ~/devel
cvs -d ~/repo checkout reactor
Al hacer esto, CVS creará el directorio reactor (es el nombre del módulo) con una copia de trabajo de todos los archivos del módulo, además de unos directorios de control llamados CVS (aparece uno en cada directorio original y no deben ser tocados). Los directorios CVS recogen el estado del módulo justo cuando se extrajo del repositorio, de manera que luego será capaz de buscar por sí mismo las diferencias adecuadas para almacenarlas. Los directorios CVS también contienen información de dónde se encuentra el repositorio, así que ya no volverá a ser necesario usar la opción *'-d'*.

CVS es una de las herramientas libres más completas para el control de versiones

Cuando se realizan proyectos entre varios programadores es posible que alguien haya modificado los fuentes del repositorio. Cada uno de los programadores puede actualizar su copia de trabajo de los módulos que tenga ejecutando el comando *'cvs update'*. Por defecto, la actualización es recursiva, es decir, en nuestro caso bastaría con ejecutar *'cvs update'* desde el directorio *~/devel* para actualizar todos los módulos que se hayan sacado del repositorio con *'cvs checkout'*. O bien, si sólo se quiere refrescar el módulo reactor, se ejecutará *'cvs update'* desde *~/devel/reactor*.
Un *'cvs update'* actualiza los archivos a la ultima versión del repositorio pero sin destruir los cambios que hayamos hecho. Es decir, mezcla los cambios en el repositorio con nuestros propios cambios e informa de forma abreviada. El informe se limita a una línea por fichero modificado (se indica con la letra U de *updated*) o en el que se han detectado cambios en la copia local (se indica con la letra M de *modified*). Nuestros cambios no se introducen automáticamente en el repositorio hasta que se validen con la orden *'cvs commit'*.
La orden *'cvs commit'* introduce todos los cambios que existan en la copia local dentro del repositorio generando, automáticamente, una nueva versión para esos ficheros. De forma automática aparecerá otra vez el editor (por

defecto, el vi) para que introduzcamos un mensaje que describa brevemente todos los cambios realizados. Este tipo de mensajes será necesario proporcionarlos para cualquier operación que suponga una modificación del repositorio.
Con los comandos de CVS descritos hasta ahora es suficiente para lograr uno de los principales beneficios de los sistemas de control de versiones, la tolerancia a errores. Si por cualquier razón se destruye cualquiera de los ficheros, un simple *'cvs update'* recupera la ultima versión. Si unas modificaciones experimentales de un fichero resultan un fracaso completo, se puede recuperar la versión del repositorio simplemente borrando el actual e invocando *'cvs update'*.
Evidentemente, esto hace más difícil eliminar un fichero realmente del proyecto:

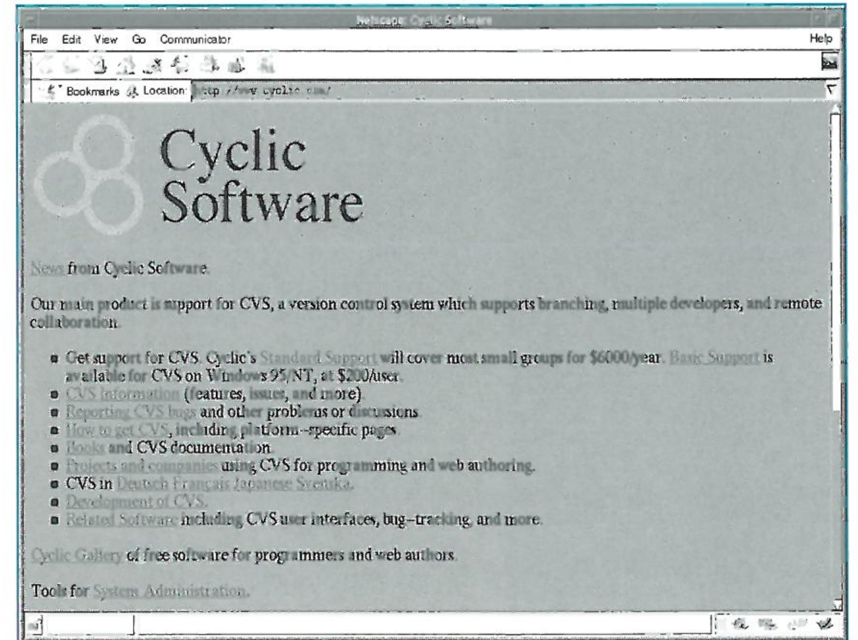
rm nombre_fichero
cvs remove nombre_fichero
cvs commit

Hay que eliminarlo primero de la forma tradicional, con rm, para que permita luego suprimirlo del repositorio usando *'cvs remove'*. Este comando lo marca para ser borrado, pero la modificación sólo se llevará a cabo en el repositorio cuando se validen los cambios con *'cvs commit'*. La adición de ficheros se realiza de forma similar con el comando *'cvs add nombre_fichero'* y los subdirectorios también son tratados de forma análoga. Al añadir un subdirectorio con *'cvs add nombre_dir'* se crea, automáticamente, el directorio CVS dentro de él con la información de control de cvs.
En cualquier momento se puede saber qué versiones de cada fichero existe en nuestra copia local utilizando el comando *'cvs status'*. El informe que genera incluye información detallada sobre fechas, versiones y estado (modificado, actualizado, etc.) de todos los ficheros del módulo. Los números de versión de cada fichero son asignados automáticamente por cvs.

ETIQUETAS Y RAMAS

Conforme evoluciona un proyecto, cada archivo tendrá un número diferente de versión. En principio esto no importa pero es poco mnemotécnico. Si por cualquier razón queremos una versión concreta de un fichero no va a ser fácil recordar los números. Para facilitar las cosas, CVS permite definir etiquetas (*tags*) que pueden ser alfanuméricas. Para esto se utiliza el comando *'cvs tag'*:

FIGURA 2.



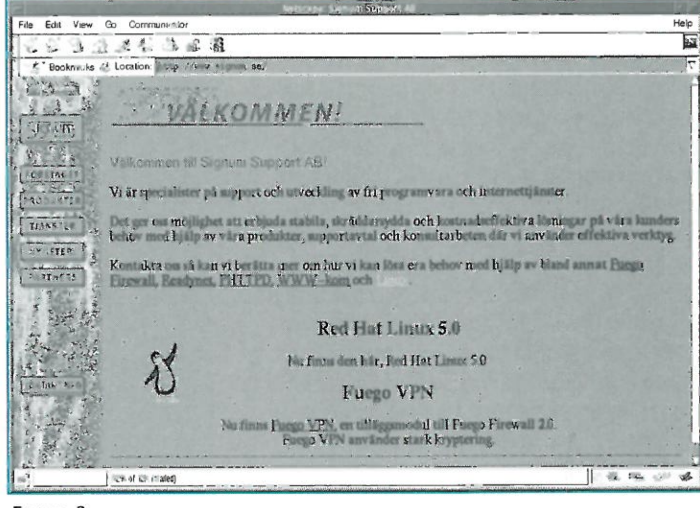


FIGURA 3.

cvs tag reactor-1_0

Por desgracia existe una limitación en los caracteres que acepta como etiqueta válida: no se puede usar casi ningún signo no alfanumérico. Un convenio bastante utilizado para las etiquetas es utilizar el mismo nombre que para las distribuciones (*releases*) pero sustituyendo los puntos por caracteres de subrayado. Por ejemplo, si la distribución se llama gcc-2.8.0 la etiqueta podría ser gcc-2_8_0. Sin embargo, no es un criterio universal puesto que el etiquetado no siempre se corresponde con una distribución.
Ahora, en cualquier momento, se puede obtener una copia local de la versión etiquetada utilizando la opción *'-r'* de *'cvs checkout'*:

cvs checkout -r reactor-1_0 reactor

Hasta ahora hemos considerado un desarrollo lineal del proyecto pero, en la vida real, esto no suele ser así. Por ejemplo, cuando se hace una distribución el código se supone estable, pero pueden encontrarse bugs posteriormente. Si las correcciones se integran sobre la última versión de desarrollo el resultado puede contener aún más bugs debidos a las últimas adiciones. Lo normal es que cuando se realiza una distribución, se ramifica también el desarrollo en una rama estable y otra inestable. En la rama estable sólo se aplican correcciones, nunca nuevas características.
Para crear una rama también se puede utilizar el comando *'cvs tag'*, pero en esta ocasión con la opción *'-b'*:

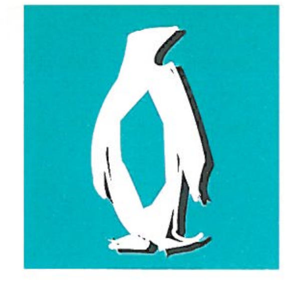
cvs tag -b reactor-1_0-patches

La línea principal de desarrollo sigue existiendo y no se mezcla con la nueva rama. Para poder trabajar en la nueva rama es preciso volverla a extraer en otro directorio con:

cvs checkout -r reactor-1_0-patches reactor

Si se trabaja en una rama (puede verse con *'cvs status -v'*) todos los cambios que se introduzcan afectarán sólo a la rama. Esto es deseable para aislar el desarrollo de las ramas, pero llega un momento en que resulta conveniente unificar los cambios de nuevo en otra rama. Es el caso, por ejemplo, de la integración de parches en la rama principal. Esto se realiza con la opción *'-j'* (de join) del comando *'cvs update'*. Por ejemplo, si estamos en la rama de desarrollo normal (reactor-1_0) y queremos incorporar los cambios de la rama de correcciones (reactor-1_0-patches) bastaría:

cvs update -j reactor-1_0-patches
cvs commit



Inmediatamente después de mezclar dos revisiones conviene poner una etiqueta en la rama que se ha integrado. Se podría hacer utilizando el comando *'cvs tag'* pero requeriría volver a sacar la rama con *'cvs checkout'*, así que es preferible usar el comando *'cvs rtag'*, similar a *'cvs tag'*, pero no necesita tener una copia local del módulo:

cvs rtag -r reactor-1_0-patches merge1 reactor

Pone la etiqueta *'merge1'* a la ultima revisión de la rama de correcciones, que es la que se ha empleado para integrar con la rama principal. Esta etiqueta permite, después, integrar cambios posteriores sin necesidad de volver a aplicar los anteriores (que podrían generar problemas como veremos más adelante). Por ejemplo, desde una copia de la rama principal:

cvs update -j merge1 -j reactor-1_0-patches
cvs commit

Utilizando dos veces la opción *'-j'* se indica el rango de versiones que se desean integrar. En este caso, se integran los cambios entre la versión *'merge1'* y la versión más reciente de la rama reactor-1_0-patches.

Más CVS

En este artículo tan solo se explora por encima las posibilidades de este programa. CVS cuenta con multitud de características relativamente avanzadas que facilitan el desarrollo de software en grupos dispersos. Por ejemplo, puede configurarse como servidor y utilizarse remotamente con varios mecanismos de autentificación; también soporta compresión con gzip en las transferencias para facilitar su uso a través de un módem. Permite que los usuarios que lo deseen sean notificados cuando se realiza alguna operación sobre el repositorio (por ejemplo, los 'cvs commit') utilizando el comando 'cvs watch'. También permite la automatización de tareas burocráticas mediante la ejecución automática de scripts ante determinadas operaciones sobre el repositorio (por ejemplo, la actualización automática del servidor de ftp o de las páginas de web de la aplicación). CVS facilita todo un mundo de nuevas posibilidades que justifican, de sobra, la existencia de dos compañías de software exclusivamente dedicadas a ofrecer soporte comercial (Cyclic y Signum).

A pesar de la sofisticación de CVS basta aprender unos pocos comandos para poder apreciar su utilidad. La mala memoria no debe ser un impedimento para usar CVS puesto que todos los comandos tienen ayuda en línea utilizando la sintaxis 'cvs -H comando', y si tampoco se recuerda esto se puede obtener una breve sinopsis con el mecanismo estándar de ayuda en línea de los programas de GNU, la opción —help ('cvs —help'). Además, CVS cuenta con amplia documentación hipertexto en formato info.

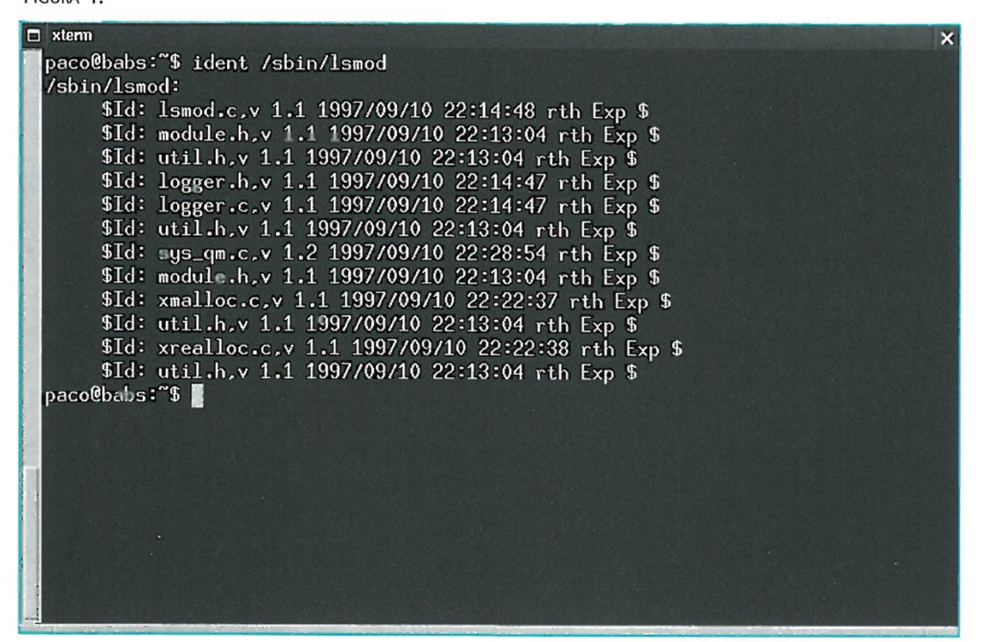
Si aún así parece difícil su uso existe un puñado de aplicaciones que actúan como interfaces intuitivos y simplificados a CVS. Por ejemplo tkcvs o pcl-cvs (para emacs). Algunos de estos interfaces serán objeto de artículos posteriores.

CONFLICTOS

Tanto con las unificaciones de ramas como con los simples *'cvs update'* realizados por múltiples personas pueden surgir situaciones en las que varias versiones pretendan cambiar un mismo conjunto de líneas de código. En ese caso, CVS no tiene datos suficientes para decidir cuál es la mejor opción y delega la decisión en el usuario, notificando que ha habido conflictos y los ficheros en los que se produjeron. Estos ficheros mostrarán, de forma muy destacada, los fragmentos que generaron el conflicto con las dos versiones contradictorias. El usuario deberá editar cada fichero para seleccionar la mejor alternativa en cada caso.

Normalmente, los conflictos son muy raros porque las personas que trabajan en un mismo proyecto no lo hacen sobre las mismas líneas de código a la vez, eso sería un caos. Sin embargo, en las unificaciones de ramas, que no es una actividad cotidiana, es relativamente frecuente encontrar conflictos.

FIGURA 4.



IDENTIFICACION

En todos los cambios del repositorio es preciso escribir un mensaje que describa brevemente la operación. Estos mensajes constituyen una especie de diario del proyecto que puede examinarse con el comando *'cvs log'*. En el informe generado por este comando aparecen todos los detalles sobre versiones, revisiones, quién y cuándo realizó cambios sobre cada fichero, etc.

Cuando se trabaja sobre una copia local, controlada por CVS, es muy fácil saber con qué versiones se está trabajando mediante los comandos *'cvs log'* y *'cvs status'*. El problema surge cuando se maneja una distribución que ya no controla CVS. Para facilitar también en estos casos la identificación CVS realiza automáticamente la expansión de ciertas variables cuando se realiza un *'cvs commit'*. Por ejemplo, una de las más usadas es la variable Id (las variables expandidas por CVS siempre van entre signos $) se expande de una forma similar a esta:

$Id: reactor.c,v 1.2 1998/02/26 01:04:16 paco Exp $

Contiene el nombre del fichero, la versión, la fecha y el autor de la ultima modificación. En cada *'cvs commit'* se actualizará el contenido de esta variable.

Una utilidad de estas variables es permitir la identificación de las versiones exactas de cada fichero de los que se construye un binario ejecutable. Para ello, basta utilizar definiciones como esta en todos los ficheros *.c

static char strid[] = "Id";

Ahora puede utilizarse la utilidad *'ident'* para identificar completamente un binario.

La expansión automática de variables es un problema cuando se trata de manejar archivos binarios, como imágenes gráficas o capturas de audio. En estos casos conviene desactivar la expansión automática indicando a CVS que se trata de ficheros binarios utilizando la opción *'-kb'* en el comando *'cvs add'*.

El único problema importante de CVS es que, actualmente, no maneja nada bien los enlaces simbólicos. Nunca deben meterse enlaces simbólicos en el repositorio.



CORREO GNU/LINUX

> **En la serie "preguntas GNU/Linux del lector" se responde a preguntas directas, que los lectores pueden enviar a la redacción, relacionadas con GNU/Linux. Estas preguntas pueden ser relativas a los artículos de la serie Foro Linux o cualquier otra consulta.**

Enlaces en un CD-ROM con Debian

El encontrar en el kiosco una revista exclusicamente para Linux [relativo a la revista Linux Actual] a los que estabamos interesados en instalarlo nos ha resultado una grata sorpresa. También ha sido una sorpresa que una vez puesto a la instalación de Debian (que se adjuntaba en un CD a la revista) resulta que cuando se indica que vayamos al directorio Debian-1.3 nos encontramos con que dicho directorio no existe (al igual que Debian-1.rxx). Sí existen unos ficheros de tamaño 0 (cero). Supongo que puede ser debido a un error en la copia del CD-ROM. Pregunto ¿Sería posible canjear el CD-ROM con la distribución por uno que no tuviese ese error?"

José A. Barcia
jbarcia@cica.es

Los CD-ROMs pueden ser grabados con muchos formatos. Uno de ellos, y quizá el más popular, es como pistas de audio; otro puede ser algún sistema de ficheros. El sistema de ficheros que más se suele utilizar con los CD-ROMs es el ISO 9660. Ahora bien, en Linux (como clon de UNIX) existen permisos en los ficheros. Esto es lo que evita que los usuarios puedan inadvertidamente borrar ficheros de otros usuarios o programas, mientras pueden ejecutar esos programas. Otra de las cosas que incorporan los sistemas de ficheros en Linux son los enlaces. Mediante un enlace puede hacerse que un fichero "apunte" a otro. Se puede, por ejemplo, hacer que un enlace apunte a un directorio, pudiéndose "entrar" en el enlace con lo cual se está entrando en el directorio al que apunta.

En el caso de una distribución Debian GNU/Linux 1.3.1 el directorio Debian-1.3.1 de la raíz del CD-ROM es en realidad un enlace que apunta a bo (nombre por el que es conocida esta distribución). Por otro lado el bo/disks-i386/current es un enlace a bo/disks-i386/1997-05-30.

En un sistema como MsDOS o Windows95 entre otras muchas carencias no se dispone de enlaces (en MsDOS tampoco se dispone de nombres largos). Por lo tanto desde uno de esos sistemas los enlaces se muestran como ficheros de longitud cero.

En la explicación del contenido del CD-ROM, se refería a ficheros en directorios que en realidad eran enlaces. En sistemas que no entiendan enlaces aparentemente no se puede acceder a ellos y se tiene que acceder directamente por el directorio al que apuntan. En ISO 9660 con extensiones RockRidge se usa el fichero TRANS.TBL para saber el tipo de fichero, el nombre largo y el destino de un enlace. Algunos sistemas operativos entienden este fichero. Si su sistema no sabe hacerlo automáticamente puede obtener esa información consultando dicho archivo.

Aplicaciones para GNU/Linux

"[...] El modo de trabajo en Linux, ¿es igual que Windows o DOS? [...] ¿Valen las aplicaciones y paquetes de trabajo (Office, Internet Explorer, Outlook, Paradox, Antivirus, etc.) para Linux? [...] ¿Exiten aplicaciones potentes para Linux?"

A. Lorente
alorente@arrakis.es

En GNU/Linux se puede trabajar de manera similar a DOS, con una shell (o intérprete de órdenes), y también similar a Windows utilizando el sistema de ventanas XWindow. En GNU/Linux, en cambio, no se fuerza a que un usuario use uno u otro (o cualquier otro). Así se puede conseguir que un "obsoleto", 386SX a 16MHz con 8 MB de RAM funcione perfectamente como un router conectando 6 redes de área local, o como servidor de Internet. WindowsNT, simplemente no puede. La sobrecarga que el sistema de ventanas, etc. impone no le permite ni tan siquiera ser instalado en ese sistema.

Sobre la segunda pregunta, esas aplicaciones no son directamente utilizables desde GNU/Linux. Existen varias estrategias para afrontar este punto:

Las compañías ofrecen versiones compiladas para GNU/Linux. Esto no es tan complicado de realizar para ellas cuando el código se ha programado con unos mínimos criterios de portabilidad. Muchas compañías (como Corel con WordPerfect y CorelDraw, o Netscape) utilizan esta alternativa.

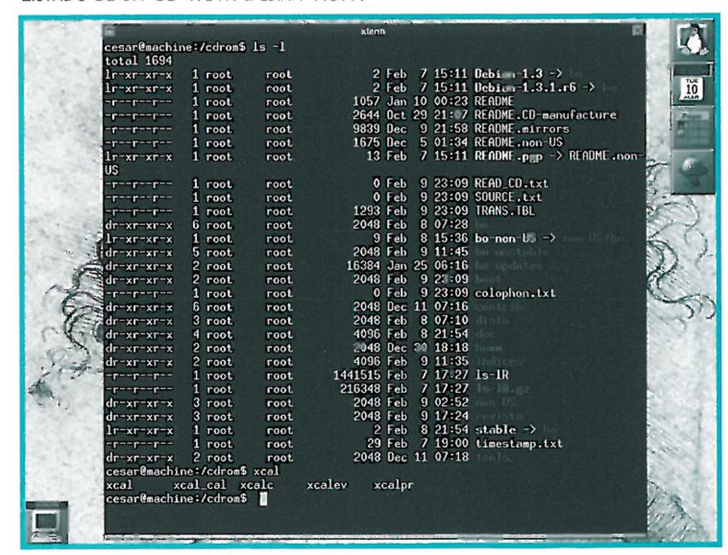
Utilizar software libre alternativo al software propietario que utilizaba. Esta es normalmente la mejor opción por varios motivos: los programas libres evolucionan para los usuarios. De hecho, en muchas ocasiones los desarrolladores son los propios usuarios expertos. Así, en los programas libres no se encontrará con que una nueva versión utiliza un formato incompatible con otras versiones para que tenga que actualizarse. Puede leer en el número 1 de "Linux Actual" y en la serie Foro Linux muchos ejemplos de las ventajas de utilizar software libre. Si no desea desprenderse del trabajo anterior puede cambiar a un programa que le permita leer el formato de ficheros que use (por ejemplo, varios programas como ApplixWare o StarOffice permiten leer y escribir ficheros doc, ...)

Utilizar un emulador, en este caso de Windows, para poder ejecutar todas estas aplicaciones. Existen dos alternativas: Wine y Wabi. Wine es software libre, y aunque aún con algunas limitaciones, permite ejecutar código de win32 y en Xwindow usando 16 bits de profundidad (65535 colores, y no sólo 256). La principal traba con que se encuentran los desarrolladores de Wine son las llamadas indocumentadas de Windows. Wabi es un producto propietario, y funciona de manera muy estable, con la salvedad de que sólo lo hace en displays de 256 colores y no permite ejecutar aplicaciones win32.

Respecto a los antivirus, no necesitará utilizarlos si usa GNU/Linux. Simplemente porque los virus para GNU/Linux no existen. En MsDOS y Windows cualquier usuario puede modificar cualquier fichero del disco, por lo tanto un virus puede copiarse y borrar a su antojo. En GNU/Linux los ficheros del sistema no pueden ser modificados por los usuarios (aunque lógicamente sí pueden usarlos), y por lo tanto los virus no pueden infectarlos o destruirlos.

Como respuesta a tu tercera pregunta, existen muchísimas aplicaciones

LISTADO DE UN CD-ROM DEBIAN 1.3.1.





Correo GNU/Linux

potentes para GNU/Linux, que abarcan la mayoría de los campos de uso de ordenadores. Existen desde aplicaciones de ofimática a programas de gráficos 3D (se ha usado GNU/Linux para los efectos especiales de la película Titanic), pasando, lógicamente, por herramientas de desarrollo y software para Internet.

Introducción a LILO

"Tengo en mi ordenador Windows95 y GNU/Linux. ¿Cómo puedo, de una manera rápida, lograr que mi ordenador me permita en el arranque elegir entre ambos sistemas? He oído hablar acerca de LILO. ¿Es difícil de instalar?"

J. J. Yubero
jjyubero@ieeesb.etsit.upm.es

Efectivamente LILO es una buena elección en estos casos. LILO significa LInux LOader. Es un programa especialmente diseñado para permitir elegir al arrancar entre varios sistemas en la misma máquina. Para configurar LILO se utiliza el fichero /etc/lilo.conf. Aunque LILO puede ser configurado con multitud de opciones (vea /usr/doc/lilo para los detalles), puede verse un ejemplo sencillo en el listado adjunto. Este ejemplo permite arrancar Windows95 y GNU/Linux. En el ejemplo se supone que ambos sistemas están instalados en el mismo disco duro, Windows95 en la primera partición y GNU/Linux en la segunda. Despues de editar /etc/lilo.conf se procede a instalar LILO en el sector de arranque (boot sector) del disco, tal como se indica en la línea "boot=", con:

```
root@mimachine:/etc# lilo
Added Linux *
Added Linux-old
Added win95
root@mimachine:/etc# _
```

Al arrancar el ordenador mostrará la palabra "LILO" en la pantalla. Tras 5 segundos (en el fichero de configuración se indican 50 décimas de segundo de espera), arrancará la entrada con el label "Linux" de /etc/lilo.conf porque era la primera. Si se desea arrancar otra entrada, se debe pulsar Shift y escribir su label. Como se ha mostrado LILO se puede configurar para elegir entre varios kernels diferentes que se tengan compilados, o entre varios sistemas operativos.

LISTADO 1. SIMPLE /ETC/LILO.CONF

```
boot=/dev/hda
root=/dev/hda2
compact
install=/boot/boot.b
map=/boot/map
vga=normal
delay=50
image=/vmlinuz
  label=Linux
  read-only
image=/vmlinuz-old
  label=Linux-old
  read-only
other=/dev/hda1
  label = win95
  table = /dev/hda
```

Uso del CD-ROM y del disquete

[...] Para usar el CDROM en Linux, ¿que tengo que hacer? ¿Es necesario ser superusuario? [...] ¿Es posible utilizar la disquetera en Linux sin ser superusuario?

Marcos Sánchez
marcos@impronta.es

GNU/Linux es un sistema operativo basado en UNIX, y como tal tiene un árbol de directorios único. No existen "unidades de disco" u otras referencias al hardware (innecesarias, por otro lado) para el usuario de la máquina. Unicamente el superusuario tiene que preocuparse de qué hardware tiene la máquina y de que este sea accesible a todos los usuarios.
En el caso del CDROM el superusuario puede configurar la máquina para que todos los usuarios tengan acceso. Los usuarios no tienen más que, como su propio nombre indica, usar este recurso. Para que el CDROM sea accesible debe estar "montado", es decir, su contenido debe formar parte del árbol de directorios. Todos los dispositivos de almacenamiento se "montan" en UNIX para poder acceder a su contenido desde el único árbol de directorios. Los discos duros también se montan,... incluso se pueden montar unidades remotas.
El superusuario debe especificar qué dispositivos deben montarse y como debe hacerse mediante el fichero /etc/fstab. El formato de este fichero puede encontrarse en la página de manual, haciendo 'man fstab'. Aquí se va a analizar una linea típica para utilizar un CDROM y permitir que los usuarios lo monten:

```
# <file system> <mount point> <type> <options>  <dump> <pass>
...
/dev/hdd      /cdrom   iso9660   ro,user,noauto  0   1
```

Cada linea del fstab tiene seis campos:
- *El primero es el directorio que apunta al dispositivo que se debe montar, en este caso /dev/hdd, porque el CDROM esta en el cuarto canal ide (hdd: hard disk d)*
- *El segundo es el directorio dónde se debe montar. Una vez montado el contenido del CDROM se verá bajo ese directorio. El directorio debe estar creado previamente.*
- *El tipo indica qué tipo de sistema de ficheros contiene el dispositivo. Los CDROM tienen el tipo ISO9660.*
- *Las opciones a continuación dicen como se debe montar. El CDROM debe montarse despues de introducirlo en la unidad lectora, y desmontarlo antes de sacarlo, puesto que no se monta la unidad lectora, sino el CDROM que está dentro y su contenido. Es por esta razón que, aunque normalmente el montaje solo lo hace root, en este caso debería poder hacerlo cualquier usuario (¡cómo vamos a molestar a root cada vez que queramos cambiar el CD!). Eso se indica con la opción user. La opción noauto especifica que el CDROM no debe montarse durante el arranque, ya que la unidad lectora podría estar vacía. Por último la opción ro (read only) indica que se debe montar para solo lectura. No se puede escribir en un CDROM.*
- *El quinto campo no se utiliza con el CDROM y por eso se pone a cero. El sexto indica si se debe comprobar la consistencia del sistema de ficheros en el montaje.*

Si el superusuario configura el fichero de esta forma los usuarios despues de introducir el CDROM en el lector deben teclear:

```
cesar@mimachine:$ mount /cdrom
cesar@mimachine:$ _
```

Y el contenido del CDROM será accesible en el directorio /cdrom. Es importante desmontarlo antes de sacarlo del lector con:

```
cesar@mimachine:$ umount /cdrom
cesar@mimachine:$ _
```

Por otro lado, Linux impedirá la apertura de la puerta de la unidad de CD-ROM si éste no esta desmontado.
La disquetera, como el CDROM, es un dispositivo que se debe montar antes de acceder a su contenido. Sería necesaria una línea equivalente a la explicada en la pregunta anterior para poder montar los disquetes. Cambiarían, sin embargo, algunas cosas como el directorio que representa este dispositivo, que es /dev/fd0, o que el montaje no se realizaría para solo lectura, y que el sistema de ficheros del disquete no es iso9660 (sino por ejemplo el de MsDOS). Pero en GNU/Linux no es necesario montar y desmontar los disquetes cada vez que se quiera leer o escribir en ellos. Es preferible utilizar un conjunto de herramientas agrupadas en el paquete mtools. Cualquier distribución Debian GNU/Linux incluye este paquete, y está también disponible en otras distribuciones. Se trata de software de libre distribución que permite acceder a los disquetes de modo similar al utilizado en MsDOS. Las órdenes que proporciona llevan el mismo nombre que sus equivalentes en MsDOS, precedido por la letra 'm': mdir, mcopy, mformat, ...
La sintaxis es la misma que en MsDOS, y la disquetera se denota con "a:" Para, por ejemplo, ver el contenido de un disquete, se utiliza:

```
cesar@mimachine:$ mdir a:
 Volume in drive A has no label
 Volume Serial Number is 227A-58E1
 Directory for A:/

drvspace bin    71559 08-24-1995  9:50a
command  com    95334 08-24-1995  9:50a
format   com    41127 08-24-1995  9:50a
sys      com    13575 08-24-1995  9:50a
autoexec bat      529 05-26-1997 12:44a
display  sys    17239 08-24-1995  9:50a
country  sys    27094 08-24-1995  9:50a
........................
........................
```

Sería un fragmento la salida del comando.

Capítulo 13º
Autor: María Jesús Recio



Utilidades TCP/IP

Para manipular con soltura estos protocolos, no basta con conocer la teoría de su funcionamiento, ni siquiera es suficiente saber el nombre y los parámetros de las aplicaciones TCP/IP más conocidas. Para administrar una red TCP/IP con seguridad es necesario manejar en profundidad algunas herramientas clave.

Hasta ahora, en los artículos de esta serie dedicados a TCP/IP se ha visto mucha de la teoría relacionada con el mundo de esta familia de protocolos: ubicación de los protocolos en el modelo de capas, formato de las tramas, direccionamiento, nombres de máquinas, DNS, HDCP, etc. Con toda esta información, el lector tiene una visión genérica del funcionamiento de estos protocolos, sin embargo, le faltan demostraciones prácticas de cómo administrar una red TCP/IP o información sobre cómo conectar la red a Internet.
Por ejemplo, cuando se ha hablado de DNS se ha explicado la arquitectura de este mecanismo de resolución de nombres, sin embargo, cuando un administrador decide conectar una red o, simplemente, un ordenador a Internet, ¿sabe qué es lo que tiene que hacer para que su máquina (nombre de *host*) aparezca en los servidores de nombres correspondientes? Al igual que ésta, existen muchas situaciones reales en las que un buen administrador debe conocer los mecanismos para solucionarlas.

NOMBRE DE MAQUINAS EN INTERNET

Cuando se dispone de una red relativamente pequeña, no conectada al mundo exterior, es decir, no conectada a Internet, el administrador puede pensar en crear el fichero */etc/hosts* a mano en todos y cada uno de los ordenadores de la red. Hoy en día esto está obsoleto: es mucho más cómodo crear en la red un servidor de nombres local.
Si en una red local ya no está justificada la generación de forma manual del fichero */etc/hosts*, imagínese en una red corporativa que se conecta a Internet. Sin embargo, el DDN NIC (*DDN Network Information Center*), aunque no de forma manual, sigue manteniendo la existencia de una tabla *hosts* para aquellos que no quieran utilizar un servidor de nombre.
Como puede imaginarse, esta tabla es enorme (teóricamente tiene el nombre y la correspondiente dirección IP de todas las máquinas de Internet), y por tanto, inimaginable introducirla a mano en un ordenador.
EL DDN NIC define dos mecanismos para el problema de la resolución de los nombres: el fichero *hosts.txt* y el servicio NDS. Cuando un administrador decide conectar su red a Internet, debe optar por una de estas dos soluciones.
El fichero *host.txt* contiene tres tipos de entradas: registros correspondientes a redes, registros correspondientes a *gateways* y, por último, registros correspondientes a *hosts*. En el cuadro 1 se puede ver el formato de estos registros.
A partir de este fichero, se construyen los ficheros */etc/hosts, /etc/networks* y */etc/gateways* descritos en artículos anteriores, aunque como puede ver, existen datos almacenados en estos ficheros que no se utilizan.
En algunos sistemas operativos, por ejemplo en Unix, existen comandos para generar automáticamente los ficheros mencionados partiendo del fichero *hosts.txt*. Si usted es un administrador de una red TCP/IP y quiere utilizar este mecanismo para la resolución de nombres, y dispone de máquinas Unix, lo que habría que hacer es lo siguiente:
Primeramente debe bajarse a su red, vía FTP, esta tabla. Para ello debe conectase al NIC o bien utilizar el comando *gettable* disponible en versiones BSD de Unix.
Cuando se obtiene este fichero, lo siguiente que debe hacer es generar los ficheros */etc/hosts, /etc/networks* y */etc/gateways*; para ello se utiliza el comando *htable*. Este comando genera los tres ficheros partiendo no sólo del fichero *hosts.txt* sino teniendo en cuenta también los ficheros *localhosts, localnetworks* y *localgateways* si estos existen.
Cuando se ejecuta este comando, lo primero que hace es copiar el contenido de los ficheros locales mencionados sin ningún tipo de conversión, sobre los ficheros correspondientes: *hosts, networks* y, a continuación, extrae del fichero *hosts.txt* los registros añadiéndoselos a los ficheros oportunos.
Es aconsejable realizar esta operación en un directorio temporal, después mover los ficheros generados a los directorios correspondientes (en este caso el directorio /etc) y, finalmente, borrar le fichero *hosts.txt.*
Si se va a utilizar DNS, realmente la operación que acaba de describirse tiene poca utilidad, ya que no se genera fichero *gateway* y los nombres de *hosts* (contenido del fichero *hosts* generado) son proporcionados por el DNS, por lo que, lo único que se utiliza es el fichero *networks*, pues el DNS no proporciona nombres de redes en sus bases de datos. En este caso es más conveniente copiar del NIC únicamente los nombres de redes y, para ello, en lugar de

Qué hacer en primer lugar

Cuando una organización decide conectarse a Internet, lo primero que debe hacer es definir los servicios que quiere utilizar: navegación por Internet a través de todas las máquinas de la organización, disponibilidad de correo electrónico, visibilidad de páginas informativas de la empresa en Internet. Dependiendo de los servicios solicitados así necesitará unas especificaciones que cumplir u otras. En cualquier caso, siempre será necesaria la presencia de un proveedor de servicios Internet que será el que nos conecte a la red de redes.
Cuando se quiere disponer de correo electrónico y/o presencia en Internet a través de páginas web es necesario, al menos es conveniente, disponer de un nombre de dominio propio que sea representativo y que identifique la organización. A continuación se expone todo lo relacionado con los nombres de dominio.



COMANDO	SINTAXIS	EJEMPLO	DEFINICION
Gettable	Gettable (host que contiene la tabla)	Gettable nic.ddn.mil	Obtiene el fichero hosts.txt
Hotable	Htable (fichero del que extrae)	Htable hosts.txt	Genera los ficheros correspondientes partiendo del fichero pasado como entrada.

SINTAXIS DE LOS COMANDOS RELACIONADOS CON LA GENERACION DE LOS FICHEROS DE CONFIGURACION DE TCP/IP.

bajarse el fichero *hosts.txt*, lo que el administrador de la red debe hacer es bajarse vía ftp el fichero *networks.txt*, también generado por el NIC. Una vez que se dispone de este fichero, basta con ejecutar el comando *htable* pasándole como parámetro el fichero *networks.txt* en lugar del fichero *hosts.txt*.
Si se decide utilizar DNS, para así disponer de las ventajas que este mecanismo incorpora sobre el descrito anteriormente, el administrador de la red debe ponerse en contacto en el NIC que es el organismo que tiene la autoridad para asignar nombres de dominio, indicándole el nombre que se ha elegido, así como, y esto es muy importante, las máquinas (dirección y nombre), al menos tienen que ser dos, que van a hacer de servidores de nombre del dominio que se solicita. Si es posible la asignación de este nombre, el NIC concede la petición. El dominio creado estará bajo algunos de los dominios de primer nivel (si recuerda lo explicado en artículos anteriores estos dominios podían ser de dos tipos, referentes al país y referentes a la actividad a la que se dedicaba la empresa, por ejemplo: eu, es para el primer modelo, y com, edu, mil para el segundo).

Cuando se obtiene la concesión de un dominio, se obtiene también la autoridad para crear todos los subdominios que se quiera por debajo de éste

Cuando el administrador obtiene la concesión de un dominio obtiene también la autoridad para crear todos los subdominios que él quiera por debajo de éste. Así, si se ha solicitado un dominio, por ejemplo ACME, para una organización de tipo comercial (caerá bajo el dominio com); una vez concedido el dominio *ACME.com* la empresa tiene autoridad para crear subdominios, por ejemplo departamentales por debajo de éste, como pueden ser ventas o markenting, por lo que podrán existir máquinas que pertenezcan a dominios como: *marketing.acme.com,* o bien *ventas.acme.com.*

Además de comunicar la concesión de un dominio, el NIC actualiza los servidores de dominio correspondientes, en este caso los servidores del dominio de primer nivel .com con el nombre del nuevo dominio estableciendo, para ello, enlaces con las máquinas definidas como servidores de nombres de este nuevo dominio. De esta forma, el nuevo dominio queda registrado y es localizable por toda la red.

ORGANISMO PARA LA CONCESION DE DOMINIOS

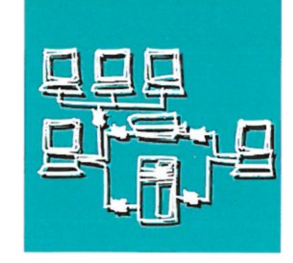
No solamente el NIC es el organismo para la concesión de nombres de dominio; existen otros organismos que también están capacitados para realizar esta labor. En realidad, depende del dominio del primero del que se trate. Por ejemplo, para el dominio de primer nivel ".es" (de España) existe un organismo especial que es el encargado de realizar las labores del NIC. Las reglas por las cuales se rige este organismo para la concesión de dominios son mucho más estrictas que las que mantiene el NIC, en cuanto a especificaciones que la empresa solicitadora del dominio debe cumplir. Además, los trámites en este caso son mucho más lentos, por lo que se tarda más tiempo en obtener un domino perteneciente al dominio de primer nivel .es que a otros. Por eso, no es de extrañar que muchas empresas españolas tengan como dominio de primer nivel .com, en lugar de .es.
Una organización puede disponer de más de un nombre de dominio, ya que puede pertenecer a más de un dominio de primer nivel. Muchas empresas solicitan un dominio bajo el dominio .com y, además, un dominio bajo el dominio .es. Primeramente obtienen el dominio .com y empiezan a funcionar con él, y más tarde obtienen el dominio bajo .es. Llegado este punto, la organización tiene varias opciones: o bien quedarse con ambos dominios, o bien liberar uno de ellos.
Además de los requisitos existen otras diferencias importantes y que se deben tener en cuenta a la hora de dirigirse a uno u otro para la petición de un nombre de dominio, entre la forma de actuar entre estos organismos encargados de dar autorización para la utilización de nombres de dominio.
Por ejemplo, cuando se solicita un nombre de dominio al NIC, el dueño de este nombre es la entidad que realiza la solicitud, mientras que si se hace al IANA es la empresa para la que se solicita (debe identificarse con una serie de documentación legal) la dueña del dominio, independientemente de la entidad que lo haya solicitado. Esto, que inicialmente no parece tener mucha importancia, puede ser el origen de problemas en determinadas situaciones.

El DDN NIC sigue manteniendo la existencia de una tabla hosts, en el fichero hots.txt

Normalmente, cuando una organización decide solicitar un nombre de dominio es porque quiere darse a conocer al mundo a través de servidores web en Internet. Para llevar esto a cabo es necesaria la intervención de una empresa proveedora de acceso a Internet, que va a ser la encargada de dar las especificaciones necesarias para la puesta en práctica de este servicio, así como la que solicite el dominio al organismo pertinente, y aquí pueden surgir los problemas.
Imagine que usted, como responsable de la puesta en marcha de este proyecto, decide contactar con una empresa proveedora de servicio Internet; esta empresa le preguntará el dominio que desea y ella será la encargada de solicitarlo. Si el dominio depende del NIC, la empresa proveedora lo solicitará en su nombre, si no se le dice lo contrario, aunque sea su organización quien lo utilice. Si, por el contrario, el dominio depende del IANA usted será el dueño del dominio solicitado.
Si con el tiempo, y por cualquier motivo, decide cambiar de proveedor piense que si el nombre de dominio que está utilizando no es suyo, dependerá de la calidad de la relación que mantenga con su empresa proveedora el que pueda o no llevarse este nombre al nuevo proveedor. Mientras que si se trata de un dominio solicitado al IANA, usted podrá disponer de él, independientemente del proveedor con el que trabaje.

CONFIGURACION DE EQUIPOS

De forma paralela a la obtención del nombre de dominio, en el caso de querer, además de poner en marcha una red



TCP/IP, la conexión de ésta a Internet, el administrador de la red debe instalar y configurar TCP/IP en cada uno de los ordenadores integrantes de la red.
El proceso de instalación dependerá tanto del sistema operativo del que se disponga como de la versión de TCP/IP que se esté instalando. Por ejemplo, mientras que la instalación de TCP/IP en una máquina con Windows 95 o Windows NT es relativamente sencilla, este mismo proceso en una máquina Unix es algo más complejo debido, principalmente, a la austeridad del sistema operativo en sí.
En cualquier caso, una vez instalada la pila de protocolos, para su configuración, el administrador debe poseer cierta información, como es la dirección del *gateway* por defecto, el protocolo de encaminamiento que se va a utilizar, la dirección del servidor de nombres, en el caso de que se utilizara, el nombre del dominio, la máscara de subred, si la red se segmenta, y la dirección de *broadcast*.
La dirección del *gateway*, por defecto, es necesaria en el caso de conectar la red a otras redes o segmentos. Se trata de la dirección IP de la máquina que va a hacer de puente, interconectando entre sí varios segmentos o redes.

Al solicitar un nombre al NIC, el dueño de éste es la entidad que lo solicita, mientras que si se hace al IANA es la empresa para la que se solicita

Es necesario que el protocolo de encaminamiento, en el caso de utilizarse, lo conozcan todos los dispositivos de la red. Si se utiliza DNS es imprescindible que todos los dispositivos de la red conozcan la dirección IP de éste, ya que para realizar cualquier petición, previamente la máquina debe conocer la dirección IP del destinatario, y es precisamente el servidor de nombres quien puede suministrársela.
Por razones de encaminamiento resulta fundamental que todos los dispositivos de la red conozcan cuál es la máscara del segmento al que pertenecen.
La dirección de *broadcast* es otro elemento importante a la hora de configurar TCP/IP en una máquina, pues es esencial que todos los dispositivos de una red tengan la misma dirección de *broadcast*.
Otro elemento fundamental que se debe conocer es la dirección IP que se va a dar a las máquinas. Como ya se explicó en un artículo anterior, la elección de un buen direccionamiento IP es muy importante y, para ello, se deben tener en cuenta factores como el número de ordenadores de la red (para seleccionar el tipo de direcciones, A, B

Consejos útiles

FTP SEGURO

Aunque en la actualidad existen muchas formas de transferir ficheros entre máquinas como, por ejemplo, el correo electrónico, sigue siendo el protocolo ftp (File Transfer Protocol), el camino por excelencia para este intercambio de información entre máquinas.

FTP es un protocolo de nivel aplicación de la pila TCP/IP utilizado para transferir ficheros entre máquinas, independientemente de la distancia que las separe. Permite la transferencia de información binaria o ASCII. En la actualidad existen incontables versiones de este protocolo para todos los sistemas operativos: desde versiones gráficas para entornos Windows hasta las versiones más austeras para ejecutar el línea de comando.

Se necesita muy poco: con disponer de un servidor de ftp en un extremo y un cliente de ftp en el otro, se puede aplicar este servicio, para la transferencia de ficheros en los dos sentidos. Cuando uno decide poner un servidor de ftp en su máquina, además de crear cuentas para usuarios particulares, dispone de una cuenta anonymous, sin password, a la que se conectan todos aquellos que no tengan una cuenta privada. Por tanto, dentro del sistema, existe una parte pública a la que se puede conectar cualquier persona, siendo esto un punto débil para la seguridad de la máquina.

En teoría, la idea de compartir parte de un sistema con usuarios extraños no tendría por qué ser peligroso: todos los usuarios compartiendo un directorio de trabajo y ficheros. En la realidad una estructura como la descrita duraría muy poco tiempo: debe definirse un directorio para cada usuario, de tal forma que ese directorio sea considerado su directorio raíz; así, el usuario no podrá desplazarse por otros directorios protegiendo, de esta forma, la integridad de los datos de cada uno. Para el directorio home de la cuenta ftp anonymous se utiliza la misma estructura, y quizás sea en esta cuenta donde más sentido tenga implantar un mecanismo como el descrito, ya que ante la complicidad del anonimato, los usuarios pueden hacer cualquier cosa al sistema. Otra cosa que debe tenerse en cuenta es que las cuentas que se crean para ftp no puedan utilizarse para establecer una conexión telnet con la máquina, es decir, que sólo sirvan para el protocolo ftp.

Existen mecanismos muy básicos para llevar a la práctica todo lo que se ha descrito en cuanto a proteger el sistema mínimamente frente a las posibles agresiones que pueden producirse a través de las cuentas ftp. Como en la mayoría de los sistemas operativos se implementan de forma similar, las explicaciones aquí se van a centrar sobre Unix,

Para evitar que una cuenta ftp sea utilizada para una conexión telnet, lo que debe hacerse es especificar en el fichero /etc/passwd, en el campo referente a la shell de presentación, /sbin/nullsh. De esta forma, la cuenta no puede utilizarse para la ejecución de shells.

Para conseguir que el directorio raíz de la cuenta coincida con el directorio home asociado a ésta, se debe especificar el directorio que se quiere ubicar al usuario de una forma especial:

Para especificar el directorio raíz, debe ponerse el path, que se considera directorio home, en el campo correspondiente del /etc/passwd del usuario la secuencia "/." dentro del directorio que se va a considerar directorio raíz, es decir, a continuación de éste. Por ejemplo, si para el usuario "prueba" se le quiere asignar como directorio home /usr/varios/grupo1/prueba y se quiere que el directorio raíz para este usuario sea grupo1, el path debería especificarse como /usr/varios/grupo1/./prueba.

Como se ha cambiado el directorio raíz de la cuenta, para ésta sólo existe en la máquina lo que exista dentro de su nuevo directorio raíz, por lo que será necesario situar en él todos aquellos comandos que el usuario va a necesitar utilizar, así como algunas aplicaciones necesarias para el correcto funcionamiento de la máquina. Teniendo esto en cuenta es necesario introducir los siguientes directorios:

bin dev etc usr lib

con un contenido mínimo, en cada uno de ellos, para que funcione. El contenido será:

bin: contiene los ficheros ls, tar ksh, compress y gzip (estos dos últimos para poder descomprimir ficheros enviados).
dev: contiene los ficheros tcp, ticotsortd, ucp y zero.
lib: se trata de un enlace al directorio ./usr/lib; su contenido son librerías necesarias, que dependerán de las funciones de las cuentas ftp.
etc: en este directorio se encuentran los ficheros group, netconfig y passwd.

Debe saber que este es un ejemplo, por ello, dependiendo de las necesidades de cada caso, será necesario introducir o eliminar ciertos ficheros de estos directorios.

Para crear estos directorios con sus contenidos lo mejor es definirlos una vez y generar un fichero tar con ellos, para tenerlos disponibles para posteriores cuentas. Si está pensando en copiarlos debe tener en cuenta que existen ficheros de dispositivos, que no pueden ser copiados con la orden cp, por lo que la mejor opción es generar un tar con ellos. La orden tar, podría ser como sigue: "tar avf fichero.tar /bin /dev /etc /lib /usr".

Una buena idea es crear un punto, como directorio raíz, común a todos los usuarios y, de esta forma, sólo será necesario disponer de una copia de estos ficheros.

Ahora sólo queda volcar este fichero tar al punto seleccionado; para ello escribir la orden: "tar xvf fichero.tar". Después de esto la estructura quedará perfectamente montada.



o C que identifica el tipo de red), el número de ordenadores que se tiene, si se va a conectar o no a Internet, para definir si se puede utilizar un direccionamiento privado (opción conveniente si se piensa conectar la red a Internet), o, por el contrario, si es indiferente el tipo de direcciones que se ponga (porque la red va a quedar aislada de Internet), si es necesario segmentarla (debido a sus dimensiones y para mejorar el rendimiento, realizar filtrado, etc) o se va a generar un único segmento en la red, y otros factores que se irán viendo poco a poco.
En el caso de segmentar la red es necesario definir el número de segmentos que se quiere y dar, de manera ordenada, las direcciones a cada máquina definiendo el segmento al que pertenece cada dispositivo de la red, ya que esto va a permitir, si se hace bien, además de mejorar el rendimiento de la red, trabajar con grupos de dispositivos (segmentos) de

Glosario de términos

LANE: Emulación de LAN. Tecnología que permite a una red ATM funcionar como un backbone de LAN. La red ATM debe proveer de soporte multicast y broadcast, mapeo de direcciones (de MAC a ATM), manejo de SVC y un formato de paquete que se pueda utilizar.

LAN SWITCH: Conmutador de alta velocidad que dirige paquetes entre segmentos del enlace de datos. La mayoría de los conmutadores LAN dirigen tráfico basado en direcciones MAC. A esta variedad de conmutadores LAN se les denomina algunas veces conmutador de trama. Son, a menudo, categorizados de acuerdo al método que utilizan para dirigir el tráfico: conmutación de paquetes con corte a través o conmutación de paquetes con almacenamiento y envío.

LAT (transporte de área local): Un protocolo de terminal virtual de red desarrollado por Digital Equipment Corporation.

LEAF INTERNETWORK: Dentro de una topología en estrella, se denomina de esta forma a una interred cuyo único acceso a otras interredes es a través de Ide, un core router.

LEN (low entry networking node): Nodo de red de entrada lenta. En SNA, un PU2.1 que soporta protocolos LU, pero cuyo CP no puede comunicar con otros nodos. Debido a que no hay sesión CP-a-CP entre un nodo LEN y su NN, el nodo LEN debe tener una imagen definida estáticamente de la red APPN.

LINK: Canal de comunicaciones de red consistente en un circuito o camino de transmisión, así como todo el equipo relacionado entre un emisor y un receptor. La mayoría de las veces se utiliza este término para referirse a un conexión WAN. Algunas veces se le llama línea o enlace de transmisión.

LOCAL TRAFFIC FILTERING (filtrado de tráfico local): Proceso por el cual un puente filtra (deshecha) tramas cuyas direcciones MAC fuente y destino están ubicadas en el mismo interfaz del puente, previniendo que el tráfico innecesario sea enviado a través del puente. Definido en el estándar IEEE 802.1.

DIRECCION MAC: Dirección estandarizada de la capa de enlace de datos que es requerida por cada puerto o dispositivo que se conecta a un LAN. Otros dispositivos en la red utilizan las direcciones para localizar puertos específicos en la red, y crear y actualizar tablas de rutas y estructuras de datos. Las direcciones MAC tienen 6 bytes de longitud y son controladas por el IEEE. Son conocidas como direcciones hardware, direcciones de capa MAC o direcciones físicas.

MAU: Unidad de enlace a un medio. Dispositivo utilizado en redes Ethernet y redes IEEE 802.3 que proporciona el interfaz ente el puerto AUI de una estación al medio común de la Ethernet. La MAU, que puede estar construida en una estación o ser un dispositivo separado, desarrolla funciones de la capa física incluyendo la conversión de dados digitales del interfaz Ethernet, detección de colisión, etc. A veces, se le llama unidad de acceso al medio o transceiver. En Token Ring, una MAU es conocida como una unidad de acceso multiestación y normalmente su abreviatura es MSAU para evitar confusión.

CONMUTACIÓN DE MENSAJES: Técnica de conmutación relacionada con la transmisión de mensajes de nodo a nodo a través de la red. El mensaje es almacenado en cada nodo hasta que un camino para reenviarlo está disponible.

NAP: Punto de acceso a la red. También se denomina así al lugar para la interconexión de proveedores de servicios Internet, en los Estado Unidos, para el intercambio de paquetes.

ANALIZADOR DE RED: Dispositivo que puede ser hardware o software que ofrece varias características para la resolución de problemas de red, incluyendo decodificación de paquetes para un protocolo específico, tests específicos de resolución de problemas preprogramados, filtrado y transmisión de paquetes, etc..

CAPA DE RED: Capa 3 del modelo de referencia OSI. Esta capa proporciona conectividad y selección del camino entre dos sistemas finales. La capa de red es la que interviene en el encaminamiento. Se corresponde con la capa de control del camino del modelo SNA.

NFS: Sistema de ficheros de red. Un conjunto de protocolos de sistema de ficheros desarrollados por SUN Microsystems que permite acceso remoto a ficheros a través de una red. Los protocolos NFS incluyen NFS, RPC, XDR (External Data Representation) y otros. Estos protocolos son una parte de una gran arquitectura a la que SUN se refiere como ONC.

OIM (Manejador de la Internet de OSI): Grupo encargado de tareas específicas en el que los protocolos de manejo de red de OSI pueden ser usados en redes TCP/IP.

ONC (Open Network Computing): Arquitectura de aplicaciones distribuidas diseñada por SUN Microsystems, actualmente controlada por un consorcio liderado por SUN. Como ya se ya dicho, los protocolo NFS son parte de ONC.

CONMUTADOR DE PAQUETES: Dispositivo WAN que encamina paquetes a través del camino más eficiente y que permite un canal de comunicaciones para ser compartido por múltiples conexiones. Formalmente llamado un IMP.

CONEXIÓN PUNTO A MULTIPUNTO: Uno de los dos tipos fundamentales de conexión. En ATM una conexión de este tipo es una conexión unidireccional en la que un único sistema final fuente (conocido como nodo raíz) conecta a múltiples sistemas finales destinos (conocidos como hojas).

CONEXIÓN PUNTO A PUNTO: Uno de los dos tipos fundamentales de conexión. En ATM una conexión punto a punto puede ser una conexión unidireccional o bidireccional entre dos sistemas finales ATM.

POLITICA DE ENCAMINAMIENTO: Esquema de encaminamiento que dirige paquetes a interfaces específicos basado en políticas configuradas por el usuario. Tales políticas pueden especificar que el tráfico enviado desde una red particular debería ser redirigido por un interfaz, mientras que todo el tráfico restante sea redirigido por otro interfaz, por ejemplo.

PPP (Protocolo punto a punto): Un sucesor de SLIP, proporciona conexiones router a router y host a red sobre circuitos síncronos y asíncronos; mientras que SLIP fue diseñado para trabajar con IP, PPP fue proyectado para trabajar con varios protocolos de la capa de red, como IP, IPX y ARA. PPP también ha construido mecanismos de seguridad, como CHAP y PAP, y confía en o depende de LCP y NCP.

REDIRECTOR: Software que intercepta peticiones para recursos dentro de una ordenador y los analiza para requerimientos de acceso remoto. Si se requiere el acceso remoto para satisfacer la petición, el redirector forma una RPC y envía la RPC al software del protocolo la capa del nivel inferior para transmitir a través de la red al nodo que puede satisfacer al petición.

RELAY: Terminología OSI para un dispositivo que conecta dos o más redes o sistemas de red. Un relay de la capa de enlace de datos es un puente, uno de la de red es un router, etc.

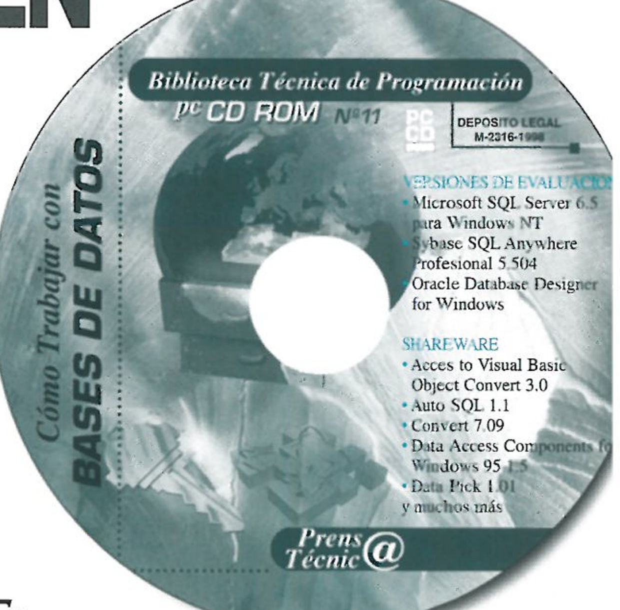
ROUTING DOMAIN: Grupo de sistemas finales y sistemas intermedios bajo el mismo conjunto de reglas administrativas. Dentro de cada dominio de encaminamiento hay una o más áreas, cada una identificada de forma única por una dirección de área.

SPANNING-TREE PROTOCOL: Protocolo puente que utiliza el algoritmo de árbol en expansión, permitiendo que un puente aprenda a trabajar dinámicamente "alrededor de loops", en una topología de red creando un árbol en expansión. Los puentes intercambian mensajes BPDU con otros puentes para detectar loops y, entonces, borran los loops reseteando los interfaces de puentes seleccionados.

CONMUTADOR: Dispositivo de red que filtra, dirige e inunda de tramas la red, basándose en direcciones destino de cada trama. Opera en la capa de enlace de datos del modelo OSI. Este término también es aplicado a un dispositivo electrónico o mecánico que permite a una conexión ser establecida como necesaria y terminada cuando no hay ninguna sesión que soportar.

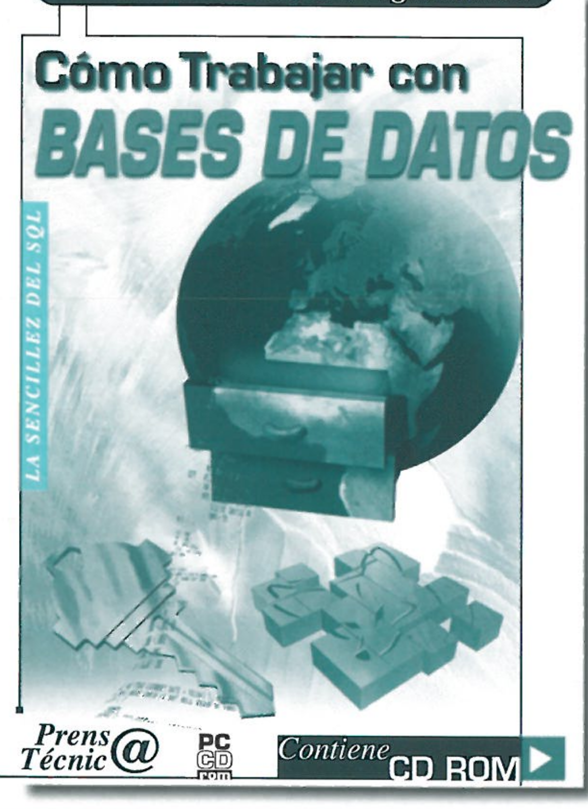
La sencillez y versatilidad del lenguaje SQL

CÓMO TRABAJAR EN BASES DE DATOS



Dentro de la tecnología informática, las técnicas de bases de datos han ido aumentando su importancia y sus aplicaciones en los últimos años. La razón de ello ha estado en la demanda, siempre creciente, de utilización de grandes masas de datos, frecuentemente integrados, y con requisitos de alta productividad en su manejo, tanto en el desarrollo de aplicaciones por programadores profesionales como por parte de usuarios finales. Las bases de datos han respondido a esta necesidad, permitiendo estructurar éstos con técnicas más formalizadas, de uso más sencillo y con mayor capacidad para reflejar su significado.

Con esta obra se pretende introducir al lector en el mundo de las bases de datos y, en concreto, en el manejo del lenguaje SQL, un sencillo pero potente instrumento para la creación, manipulación y consulta de una base de datos.



INCLUYE:

- DISEÑO DE BASES DE DATOS
- ETAPAS DEL DISEÑO
- MODELO CONCEPTUAL DE DATOS
- DISEÑO LÓGICO
- DISEÑO FÍSICO E ÍNDICES
- INTEGRIDAD REFERENCIAL
- LAS FILAS
- EL LENGUAJE SQL
- TIPOS DE SENTENCIAS SQL
- EJECUCIÓN DE LAS SENTENCIAS SQL
- CONSULTAS EN SQL
- CONSULTAS SENCILLAS
- SENTENCIA SELECT SENCILLA
- LAS EXPRESIONES
- LOS PREDICADOS
- CONSULTAS CON AGRUPAMIENTO DE FILAS
- CONSULTAS SOBRE VARIAS TABLAS
- ORDEN DE EJECUCIÓN DE LAS CLÁUSULAS DE UNA SENTENCIA SELECT
- SENTENCIA SELECT EN FORMATO COMPLETO
- SENTENCIAS SQL EN PROGRAMAS
- MANIPULACIÓN DE DATOS SIN USAR CURSORES
- LOS CURSORES
- ASPECTOS ADICIONALES DE SQL
- LAS VISTAS
- LAS AUTORIZACIONES
- NORMAS DE USO DE SQL
- INCLUSIÓN DE SENTENCIAS SQL EN PROGRAMAS
- SQL DINÁMICO

CONTENIDO DEL CD-ROM

VERSIONES DE EVALUACIÓN:

- **Microsoft SQL Server 6.5 para Windows NT**
  Microsoft SQL Server administra bases de datos en sistemas de Windows NT.
- **Sybase SQL Anywhere Professional 5.504**
  Sybase SQL Anywhere ofrece la funcionalidad de un servidor UNIX pero con menos requerimiento de software.
- **Oracle Database Designer for Windows**
  Oracle Database Designer es una utilidad gráfica para el diseño, creación y mantenimiento de bases de datos.

SHAREWARE:

- Access to Visual Basic Object Converter 3.0
- AutoSQL 1.1
- Convert 7.09
- Data Access Components for Windows 95 1.5
- Data Pick 1.01
- DB-HTML Converter Pro 1.0
- DBF Structure Fixer 2.05
- Delta 97 3.1.0
- Dynamic SQL Debugger 1.7
- Form To VB Export Wizard 1.3
- GS-Base 3.5
- HJ-TreePad 1.6
- HotSQL 1.2
- Y mucho más.



Solicite su ejemplar enviando este cupón por correo, por Fax: (91) 304.17.97 o llamando al teléfono (91) 304.06.22 de 9:00 a 19:00 h.



















Edita: Prens@ Técnic@

Nombre y apellidos ....
Domicilio .... Población ....
Provincia .... CP ....
Fecha de nacimiento .... DNI/NIF ....
e.mail .... Teléfono ....

FORMA DE PAGO

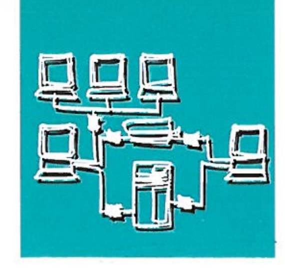
☐ Talón a PRENSA TÉCNICA ☐ Contra-reembolso Firma,
☐ Giro postal nº .... de fecha ....
☐ Tarjeta de crédito ☐ VISA nº
AMERICAN EXPRESS nº ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐
☐ Fecha de caducidad de la tarjeta ....
☐ Nombre del titular, si es distinto

☐ CÓMO TRABAJAR CON BASES DE DATOS
☐ **1.** CÓMO PROGRAMAR TUS PROPIOS JUEGOS
☐ **2.** CÓMO PROGRAMAR EN ENSAMBLADOR
☐ **3.** CÓMO TRABAJAR CON LINUX
☐ **4.** CÓMO PROGRAMAR EN VISUAL BASIC
☐ **5.** CÓMO PROGRAMAR EN LENGUAJE C
☐ **6.** CÓMO PROGRAMAR EN JAVA
☐ **7.** PROGRAMACIÓN GRÁFICA PARA PC
☐ **8.** CÓMO PROGRAMAR DELPHI 3.0
☐ **9.** CÓMO PROGRAMAR EN C++

Deseo que me envíen:

☐ UN LIBRO POR SÓLO 2995+450 ptas. gastos de envío
☐ DOS LIBROS POR SÓLO 4995+500 ptas. gastos de envío
☐ TRES LIBROS POR SÓLO 6995+500 ptas. gastos de envío
☐ CUATRO LIBROS POR SÓLO 8995+500 ptas. gastos de envío

Rellena este cupón y envíalo a: PRENSA TÉCNICA C/ Alfonso Gómez 42 Nave 1-1-2 28037 Madrid.



manera común, facilitando ciertas tareas de administración.
Cuando el administrador dispone de todos estos datos puede empezar a instalar y configurar la pila TCP/IP en todos y cada uno de los dispositivos que integran la red.

Una organización puede disponer de más de un nombre de dominio

Pero no ha terminado aquí la puesta en marcha de una red TCP/IP, y mucho menos el funcionamiento de la misma. Todavía quedan muchas tareas adicionales que un buen administrador de TCP/IP debe conocer y realizar. Además, en la mayoría de los casos, lo que se ha descrito hasta ahora queda en el nivel más teórico a pesar del punto de vista práctico que se le ha dado; siempre suceden cosas inesperadas tales como ordenadores que no se ven, direcciones IP que no quiere una máquina, segmentos lógicos no coincidentes con la arquitectura física de la red, configuración equivocada de los *gateways* con el consiguiente aislamiento de dispositivos en la red, etc.; y todo esto sin contar con la configuración de las tarjetas de red en los equipos, tema que como ya se ha tratado en artículos anteriores se va a obviar en este.

COMANDOS TCP/IP

Para que el lector se vaya familiarizando con algunos comandos y protocolos de nivel aplicación de TCP/IP, en este artículo se presenta una versión muy completa del protocolo, y también aplicación, ftp. Se trata de la versión incorporada por Windows NT 4.0. Obviamente, no todos los subcomandos presentados aquí tienen que estar presentes en todas las versiones de fpt. Se ha elegido esta versión de ftp porque, como ya se ha dicho, es muy completa, incorporando un gran número de subcomandos. La tabla 1 muestra la mayoría de los subcomandos de esta versión de ftp.

PROXIMO NUMERO

Todavía quedan muchos puntos que tratar sobre la puesta en marcha de una red TCP/IP: faltan elementos que configurar, comandos que explicar, pasos a dar a la hora de resolver determinados problemas que pueden presentarse en el día a día de una red TCP/IP, etc. En el próximo artículo se seguirá explicando todo esto y algunas cosas más.

TABLA 1. APLICACION FTP

SUBCOMANDO	UTILIDAD
!	Ejecuta el comando especificado a continuación en el ordenador local.
?	Al igual que el comando help, muestra los comandos de ftp.
Append	Añade un fichero de la máquina local al final de uno de la máquina remota. La transferencia del fichero se llevará a cabo en el modo que esté actualmente seleccionado, ASCII o binario.
Ascii	Indica que la transferencia de ficheros se realizará en modo ASCII. Es el modo predeterminado.
Bell	Activa o desactiva la opción de emitir un sonido al final de la ejecución de cada comando.
Binary	Indica que la transferencia de ficheros se realizará en modo binario.
Bye	Termina la sesión de ftp con el ordenador remoto y cierra la utilidad ftp.
Cd	Cambia de directorio en la computadora remota.
Close	Al igual que bye, sale termina la sesión de ftp y cierra la utilidad de ftp.
Delete	Borra ficheros en el ordenador remoto.
Dir	Lista por pantalla, de manera detallada, los ficheros contenidos en el directorio actual en la máquina remota.
Disconnect	Cierra la sesión de ftp con la máquina remota pero mantiene abierta la utilidad de ftp.
get	Copia una fichero, en el modo seleccionado (ASCII o binario), desde la máquina remota a la local.
glob	Activa o desactiva la utilización de comodines en los nombres de ficheros y rutas locales. Por defecto, se encuentra activado.
Hash	Por defecto se encuentra desactivado, y su activación provoca que se imprima en pantalla una marca (#) por cada 2048 bytes transferidos.
Help	Igual que la opción "?".
Lcd	Cambia de directorio en la máquina local. Cuando se inicia una sesión ftp, el directorio de trabajo de la máquina local es en el que el usuario se encontraba al iniciar dicha sesión.
Ls	Lista por pantalla, de manera simple, los ficheros contenidos en el directorio actual en la máquina remota.
Mdelete	Borra ficheros de la máquina remota.
Mdir	Lista por pantalla los ficheros de la máquina remota, pudiéndose especificar varios ficheros.
Mget	Copia ficheros desde la máquina remota a la local, utilizando el modo de transferencia especificado, ASCII o binario.
Mkdir	Crea un directorio en la máquina remota.
Mls	Lista por pantalla, de forma abreviada, los ficheros del directorio actual de la máquina remota.
Mput	Copia ficheros desde la máquina local a la remota, utilizando el modo de transferencia especificado, ASCII o binario.
Open	Conecta con el servidor de ftp que se especifique, una vez ejecutada la utilidad de ftp o después de haber ejecutado la orden disconnect.
Prompt	Activa o desactiva el modo interactivo de la transferencia de ficheros. Por ejemplo, cuando se utilizan lo comandos mput o mget se pide al usuario que confirme las copias de ficheros. Estas peticiones de confirmación pueden desactivarse con el comando prompt.
Put	Copia un fichero desde la máquina local a la remota, utilizando el modo de transferencia especificado, ASCII o binario.
Pwd	Muestra el directorio de trabajo actual en la máquina remota.
Quit	Igual que bye, finaliza la sesión de ftp y cierra la utilidad ftp.
Quote	Envía un comando ftp.
Recv	Igual que get, copia un fichero de la máquina remota a la local en el modo de transferencia especificado.
Remotehelp	Muestra información de ayuda de los comandos remotos.
Rename	Cambia el nombre de fichero en la máquina remota.
Rmdir	Borra un directorio de la máquina remota.
send	Igual que put, copia un fichero de la máquina local a la remota.
status	Muestra por pantalla el estado actual de las conexiones ftp.
trace	Activa o desactiva la opción de mostrar la ruta seguida por los paquetes cuando se ejecuta un comando de ftp.
type	Define o muestra el modo en el que se realizará la transferencia de ficheros, ASCII o binario.
user	Indica el nombre del usuario con el que se quiere establecer la sesión de ftp. De esta forma no hace falta salir de la utilidad de ftp para cambiar de usuario ni cerrar la conexión con el servidor.
verbose	Por defecto activado, muestra por pantalla las respuestas de ftp al realizar transferencias de ficheros.



El estándar set: secure electronic transactions

Desde hace más de año y medio resuena un término cada vez que se habla de comercio electrónico: SET. El punto de mira de esta sección se centrará este mes en el que promete se el "estándar definitivo" en medios de pago a través de Internet.

A lo largo de los últimos meses se ha ofrecido al lector de Programación Actual una panorámica muy extensa y bastante exhaustiva sobre el mundo del comercio electrónico, cubriendo desde problemas criptográficos hasta la mayor parte de los sistemas de pago electrónicos desde el efectivo y el cheque digitales a los soportes a pagos a través de tarjeta. Más de un lector se habrá extrañado de no ver en estas páginas al gran ausente, el estandar definitivo del que todo el mundo habla: SET.

Se ha dejado intencionadamente este tema para el final con el objeto de que el lector disponga del suficiente conocimiento de otros sistemas y medios de pago para juzgar por sí mismo si SET es la panacea universal como se trata de vender, si realmente es tan novedoso o destaca por encima de otros sistemas expuestos.

INTRODUCCION A SET

Aunque la venta electrónica no precisa de pago a través de tarjeta, la mayor parte de las ventas por catálogo o a distancia se pagan a través de las mismas en lugar de hacerse mediante cheque o contra reembolso. Por ello, la mayoría de los bancos tienen previsto proporcionar autorizaciones de tarjetas a través de Internet.

La mayor parte de las ventas por catálogo se pagan a través de tarjeta de crédito

SET, Secure Electronic Transaction es una especificación abierta y gratuita diseñada para hacer posibles los pagos con tarjeta de crédito de forma segura a través de cualquier red como Internet. La especificación ha sido desarrollada por Visa y MasterCard, con participación de otras empresas como GTE, IBM, Microsoft, Netscape, RSA, SAIC, Terisa y VeriSign.

Sus orígenes se remontan a desarrollos por separado de Visa y MasterCard en 1995. Al poco tiempo, ambas se dieron cuenta de que, en este terreno la competencia no les interesaba. Se pretendía buscar un estándar que evitara, en un futuro cercano, los costes asociados de la adaptación de estándares de facto pero, lo que es más importante, debía conservar la estructura clásica de pagos a través de tarjeta, con el fin de no dañar lo más mínimo el negocio de sus creadores. Por ello decidieron unir sus fuerzas en 1996.

Una de las primeras cuestiones que es necesario aclarar es que SET no es un nuevo medio de pago electrónico, se trata, simplemente, de un *wrapper*, una "envoltura", que hace uso de la infraestructura existente de pago con tarjetas de crédito a través de una red abierta como Internet. Así, sus objetivos fundamentales son proporcionar canales seguros entre los diversos actores, garantizar la mutua confianza a través del uso de certficados digitales y garantizar la privacidad, es decir, que la información de las transacciones sólo se encontrará disponible para las partes intervinientes cuando y como sea necesario. El objetivo no es lograr la integridad y confidencialidad de los pedidos, este tema se deja en abierto para su resolución mediante otras técnicas, como SSL. Con SET, lo que se pretende es proteger únicamente la confidencialidad de los medios de pago.

AMBITO DE LA ESPECIFICACION

Como se puede deducir de los diferentes medios de pago electrónico no efectivo que se vieron en [Echeva-1], el proceso de compra electrónica se podría dividir en nueve pasos fundamentales:

- El comprador "hojea" la oferta disponible. Típicamente se realiza a través de WWW, CD-ROM, o un catálogo en papel.
- El comprador selecciona un item (bien o servicio).
- Se presenta la factura proforma al cliente. Esta factura, que puede ser creada loalmente o enviada por el comerciante, contiene la lista de items seleccionados así como su precio, y coste total (incluyendo gastos de envío y manipulación e impuestos).
- El cliente selecciona los medios de pago. En el caso que nos ocupa, se trataría de pagos mediante tarjeta de crédito.
- El cliente envía la orden de compra al comercio.
- A la recepción de la orden de compra, el comerciante solicita autorización del pago a la red de tarjetas.
- El comerciante envía confirmación de haber recibido la orden de pago.
- El comerciante envía la información solicitada, distribuye los bienes o realiza los servicios.
- El comerciante reclama el pago al emisor de la tarjeta

SET no especifica un entorno completo de comercio electrónico, ni siquiera interviene en los nueve pasos correspondientes a un proceso de compra electrónica, sino que se limita definir los protocolos necesarios para el uso de tarjetas de crédito (fundamentalmente los pasos 5 al 9). Así, especifica la aplicación de algoritmos criptográficos (RSA y DES) y define formatos de los mensajes de certificación, compra y autorización y los protocolos entre participantes. Un punto interesante es que distingue claramente el paso 6 del 9, ya que no es lo mismo solicitar autorización para un número de tarjeta que reclamar el pago de la operación.

SET garantiza la seguridad de los pagos con tarjeta en una red abierta como Internet

SET, en cambio, no interviene en los protocolos empleados para realizar la oferta, negociación, reparto de bientes, etc, ni el contenido o presentación de los formularios, ni en médios de pago que no se basen en la utilización de tarjetas de crédito, ni en la protección de los sistemas del usuario, comerciante y pasarela de pago ante posibles ataques.



Escenario conceptual de set: actores

La especificación controla las comunicaciones entre cinco actores principales a los que denomina:
- **Cardholder: El titular del medio de pago (tarjeta).**
- **Issuer: El banco emisor de la tarjeta de crédito. Es el banco en el que se encuentra la cuenta del titular, que emite la tarjeta y garantiza el pago de las transacciones autorizadas. No se debe confundir esta entidad financiera con la marca de la tarjeta. Por ejemplo, si el banco A y el banco B pueden emitir tarjetas Visa, los emisores serían los bancos, no Visa.**
- **Merchant: El comercio que oferta bienes y/o servicios y tiene relación con un banco adquiriente.**
- **Acquirer: Banco adquiriente que lleva la cuenta del comerciante y procesa autorizaciones y pagos con tarjeta.**
- **Payment Gateway: Dispositivo operado por la entidad adquiriente o una tercera parte designada que procesa los mensajes de pago del comerciante (entre ellas las de pago del titular). Actua como pasarela de pago entre la red abierta (por ejemplo, Internet) y la red financiera privada.**

Además de estos actores, SET contempla la existencia de terceras partes, como son:
- **Brands: Instituciones financieras con marcas de tarjeta que establecen reglas y combinan roles de emisor y adquiriente, como Visa y Mastercard.**
- **Certification Authorities: Autoridades de certificación que avalan a los participantes.**

SET no especifica los medios que una entidad financiera utiliza para autenticar un titular o comerciante, cada marca puede utilizar el método que prefiera.
En adelante, se empleará terminología en castellano, aunque conviene conocer la inglesa para consultar la especificación que se incluye en el CD-ROM.

ELEMENTOS SOFTWARE SET

El sistema SET descansa sobre tres pilares fundamentales que se encuentran en la red abierta considera insegura: el comprador, el comercio y la pasarela de pago con la red segura. Será necesaria la existencia de software específico en estos tres puntos:

1. Comprador

Deberá contar con una cartera o monedero digital (*wallet*). Este es el software que almacena la información personal: certificado digital, números de tarjeta, fecha de caducidad, direcciones de envío y facturación, etc. Al igual que ocurre en el mundo físico, será necesario acudir a ella durante el proceso de compra aunque, esta vez, su acceso se encontrará protegido mediante password. Generalmente se tratará de un *plug-in* para un browser de WWW y existen dos formas de conseguirlo: 1. A través del banco emisor de la tarjeta o de un vendedor SET y 2. Puede formar parte del propio browser.

El cliente debe contar con un certificado digital prueba de que su identificador ha sido validado por el banco emisor de la tarjeta lo que implica que es el propietario legítimo de la misma. Este certificado equivale a la firma que se pone en el dorso de la tarjeta de crédito. Durante el proceso de instalación o la primera vez que se utiliza, el monedero genera una clave de encriptación y la envía al banco para que le de un visado. El banco necesita confirmar la identidad del usuario por lo que le hace algunas preguntas. Tras su verificación devuelve el certificado digital para su almacenamiento en el software monedero. Todo esto sucede automáticamente según un protocolo que forma parte de la especificación y se explica más adelante. La ventaja es que no interviene ninguna nueva compañía intermediaria, se trata directamente con un banco con el que ya se tenía relación.

SET es un sistema diseñado para mantener la infraestructura clásica de los pagos con tarjeta

2. Comercio

El comerciante deberá a su vez disponer de software SET que soporte un front-end WWW para la oferta de bienes y servicios a la vez que almacena los certificados digitales del comerciante y procesa las transacciones SET. Este software recibe el nombre genérico de *SET-compliant merchant server* (Servidor de comercio compatible SET) o "POS" (*Point Of Sale* server, servidor punto de venta). El POS es un motor de cumplimentación de transacciones que procesa las transacciones del titular, se comunica con el banco o el emisor de tarjetas y gestiona los certificados del comerciante. El comercio deberá, también, contar con un certificado digital que le acredite ante los compradores y ante la pasarela de pago. Este certificado de cara al comprador es equivalente a la pegatina con la imagen de las tarjetas de crédito que aceptan los comercios "físicos" y garantiza que el comercio tiene una relación con una entidad financiera que le permite aceptar la marca de la tarjeta de crédito.

3. Pasarela de pago

Se trata de un servidor del banco adquiriente o de la compañía de tarjetas que enlaza las transacciones SET con el sistema de transacciones establecido por la organización. Tras producirse la autorización, devuelve la información en formato SET y la envía al POS.

ELEMENTOS CRIPTOGRAFICOS EN SET

La idea fundamental de SET es apoyarse en certificados digitales para autenticar los actores participantes en una transacción. De esta forma se puede lograr, como mínimo, el mismo nivel de confianza que en cualquier otra entidad que acepte Visa o MasterCard. SET, además, hace uso de la criptografía con el fin de proporcionar confidencialidad de la información, garantizar la integridad de los pagos y autenticar a los actores involucrados en las transacciones. En este sentido, aporta, además, un elemento innovador: el concepto de *firma dual*.

1. Certificados Digitales

Como se vió en [Echeva-2], los certificados digitales se emplean para garantizar la autenticación de las partes y la autenticidad de las firmas digitales, es decir, para garantizar que las claves públicas corresponden a quien lo reivindica. Cada certificado depende de la firma de la entidad que lo firmó digitalmente. Siguiendo la jerarquía de confianza de firmas hasta una autoridad de certificación conocida y confiable, se puede estar seguro de que el certificado es válido. Por ejemplo, el certificado del titular de una tarjeta puede estar firmado por el banco emisor de la tarjeta, cuya firma estará autenticada por un certificado firmado por la marca de la tarjeta cuya firma, a su vez, estará autenticada por un certificado firmado por una raíz cuya firma es conocida por todo el software SET y que se certifica a sí misma.

La jerarquía puede contar con distinto número de niveles, por ejemplo, el certificado del titular puede estar firmado directamente por la marca de la tarjeta, que si es Visa o MasterCard, no necesitarán posterior autenticación ya que también son consideradas autocertificadas por el software SET.

En la práctica, para evitar que haya que realizar múltiples búsquedas de certificados, todos los mensajes SET incluyen todos los certificados necesarios para la autenticación de las partes a menos que se sepa que el receptor ya los tiene. Todas las partes intervinientes (titular de la tarjeta, comercio, pasarela de pago, banco emisor y banco adquiriente) deben disponer

TABLA. ELEMENTOS CRIPTOGRAFICOS EN SET

Confidencialidad	Encriptación de mensajes
Integridad	Firmas digitales
Autenticación	Firmas y certificados digitales



de sus correspondientes certificados digitales para poder envíar y recibir mensajes SET. El titular dispone, además, de un certificado diferente por cada cuenta bancaria, pero cada cuenta puede disponer de más de un certificado. De esta forma, el usuario puede instalar certificados diferentes para el ordenador de su casa, el del trabajo o cualquier otro dispositivo hardware susceptible de alojar un monedero SET.

2. Protección de comunicaciones

A lo largo del artículo se describirá una serie de comunicaciones entre partes. Cada parte interviniente dispone de un par de pares de claves asimétricas RSA, uno para el intercambio de claves simétricas y el otro para firmar digitalmenente. Las claves públicas Kp de estos pares, se encuentran avaladas mediante el certificado digital de una autoridad de certificación, C=Ksca[Kp].

La mayor parte de los mensajes van firmados digitalmente salvo, en algunas ocasiones, los de inicio de un proceso. Estas firmas digitales se realizan del modo habitual [Echeva-3], extrayendo una huella digital (digest) del mensaje (M) a través de una función one-way (garantía de integridad) y cifrándola mediante la clave secreta (garantía de autenticidad) del remitente, Ksr: F=Ksr[H(M)]. Para la extracción de huellas digitales, SET utiliza el algoritmo SHA de 160 bits.

Gran parte de los mensajes también se transmiten cifrados (garantía de confidencialidad). Para ello, se emplean claves simétricas, K, DES de 56 bits. Los emisores de mensajes generan estas claves aleatoriamente y empaquetan el mensaje, su firma digital (F) y el certificado de su clave pública (C) cifrándolo todo con la clave simétrica: M'=K[M,F,C]. Para la distribución de las claves simétricas, se vuelve a emplear criptografía asimétrica RSA, cifrando la clave simétrica con la clave pública del destinatario, Kpd: K'=Kpd[K]. Así, el mensaje enviado a destino será un par (M',K'). No se establece el concepto de sesión para las claves simétricas, o lo que es lo mismo, cada sesión equivale a un único mensaje, ya que se emplea una clave simétrica distinta para cada mensaje.

El lector debe tener en cuenta que se ha tratado de presentar aquí el mecanismo general, con el objeto de omitir todos estos detalles en el resto de las explicaciones. En algunos casos concretos se emplearán dos claves simétricas para un mismo mensaje y, cuando los titulares no dispongan de certificados, acompañarán sus mensajes únicamente de la huella digital (garantizando, por tanto, su integridad, pero no su autenticidad).

3. Firma Dual

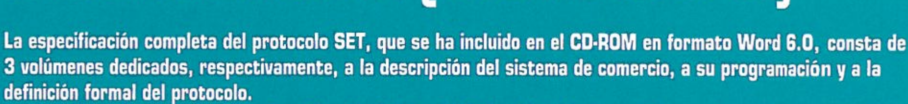
El único punto novedoso que introduce el estándar SET es el concepto de firma dual, como una nueva aplicación de las firmas digitales que permite relacionar una orden de compra (OI, *Order Instructions*) con unas instrucciones de pago (PI, *Payment Instructions*) sin que cada una de las partes encargadas de procesar una u otra necesiten (ni puedan) acceder a la información de la otra.

La idea consiste en permitir que el usuario envíe un mensaje de compra a un comercio que incluya las instrucciones de pago necesarias para el banco. El comercio deberá ser capaz de acceder a la orden de compra, pero no a la información contenida en las instrucciones de pago. Del mismo modo, el banco deberá ser capaz de acceder a las instrucciones de pago, pero no a la información sobre la compra. Pero tanto el comercio (para evitar engaños del titular), como el banco (para evitar engaños del comercio) deben ser capaces de relacionar ambas para comprobar que la orden de compra corresponde, en efecto a las instrucciones de pago.

Para ello, se lleva a cabo el mecanismo de firma dual (figura 1) según el cual, se obtienen las huellas digitales tanto de la OI como de la PI (x.a). A continuación, se concatenan ambas (x.b) y se extrae la huella conjunta, Hc, (x.c). Por último, se encripta esta huella (x.d) con la clave secreta, Kst, del firmante (el titular).

Al comercio se le permite el acceso a (OI, H(PI), Kst[Hc]). Para comprobar la correspondencia entre OI y PI le basta calcular H(OI) y extraer la huella conjunta de H(OI) y H(PI) que deberá coincir con Hc.

Análogamente, el banco tiene acceso a (PI, H(OI), Kst[Hc]) y para comprobar la relación le basta calcular H(PI) y extraer la huella conjunta de H(OI) y H(PI) que deberá coincir con Hc.



La especificación completa del protocolo SET, que se ha incluido en el CD-ROM en formato Word 6.0, consta de 3 volúmenes dedicados, respectivamente, a la descripción del sistema de comercio, a su programación y a la definición formal del protocolo.

En el primero de ellos, *SET Bussines Description* (set_bk1.zip, 80 páginas), se dispone de una información de conjunto del sistema que incluye los flujos de proceso SET.

El segundo de ellos, *SET Programmer's Guide* (set_bk2.zip, 619 páginas), contiene las especificaciones técnicas del protocolo, que incluyen todo lo que una implementación software debe incorporar para implementar completamente la especificación así como detalles sobre campos de mensaje opciones, interfaces con WWW y E-mail y proceso y recuperación de errores. Este será el volumen necesario si se desea realizar una implementación SET completa, ya que en él se encuentra toda la información necesaria para generar software tanto para titulares de tarjetas como para comerciantes.

Por último, el tercero de los volúmenes, *SET Protocol Description* (set_bk3.zip, 245 páginas), contiene la definición formal del protocolo, por lo que se aconseja únicamente a criptoanalistas interesados en realizar un estudio de la seguridad del sistema y a programadores de sistemas que vayan a desarrollar primitivas de comunicación con la finalidad de realizar pruebas.

PROCESO GENERAL DE UNA TRANSACCION SET

A continuación se describe el flujo general de una transacción SET correspondiente a la figura 2. En los apartados siguientes se entrará en detalles.

Una transacción SET comienza cuando el titular de una tarjeta de crédito envía una solicitud de inicio al comerciante.

El comerciante responde con su certificado digital, que autentica su clave pública de firma y la clave pública de la pasarela de pago.

La negociación sobre los bienes o servicios ofertados y demandados se realiza de forma externa al protocolo. Se vuelve a entrar en el ámbito de la especificación cuando el cliente

FIGURA 1. MECANISMO DE FIRMA DUAL.

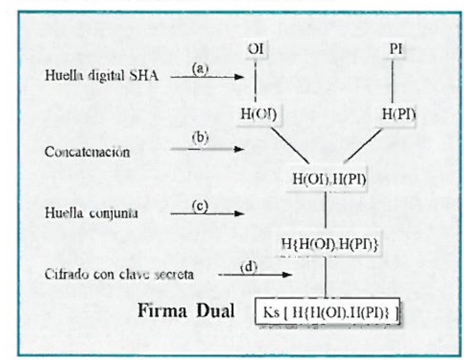
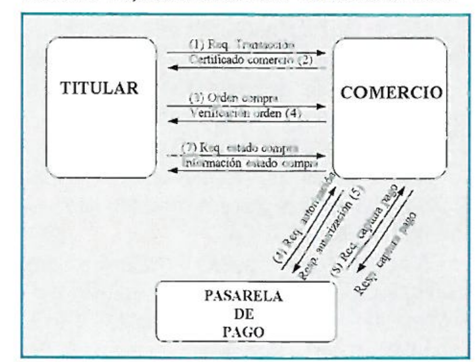


FIGURA 2. FLUJO GENERAL DE UNA TRANSACCION SET.





genera la orden de compra. Esta orden de compra lleva una firma dual y consta de dos partes: una instrucción de compra requerida por el comerciante (OI) y una instrucción de pago requerida por la pasarela de pago (PI). Esta instrucción únicamente será "visible" para la pasarela ya que el cliente la encripta con la clave pública de la misma. De esta forma, el comerciante no tiene acceso a la información de pago, como el número de tarjeta, lo que, en el sistema tradicional es la primera fuente de fraude.

El comerciante responde con la verificación de la orden de de compra.

Cuando el comerciante recibe la petición de compra genera un mensaje de autorización de pago para la pasarela. Este mensaje consta de dos partes: el mensaje de pago original, enviado por el titular en la solicitud de compra y el mensaje de autorización, generado por el comercio.

Una vez recibido este mensaje, la pasarela de pago envía la solicitud de autorización a la red financiera, generalmente privada, y tras la pertinente confirmación responde al comerciante con una respuesta de autorización.

El paso 4 puede realizarse antes o después de los pasos 5 y 6. Además, SET contempla otros mensajes que se estudiarán más adelante (7 y 8).

El proceso completo de pago SET comprende cinco etapas: 1. Registro del titular, 2. Registro del comercio, 3. Solicitud de pago, 4. Autorización del pago y 5. Captura del pago, que se presentan a continuación.

REGISTRO DEL TITULAR

El software de los titulares debe registrarse con una autoridad de certificación (CA) como paso previo al envío de mensajes SET a los comerciantes. Pero antes de enviar su información a la CA, debe conocer la clave pública de ésta con objeto de encriptarla y que la comunicación sea segura. Para ello no se fía ni de la CA, por lo que la comunicación comienza con un mensaje de solicitud de inicio en el que se solicita el certificado de la CA, certificado que contendrá las claves públicas de intercambio y encriptación de la misma firmadas por otra CA de nivel superior en la jerarquía de confianza.

El software también necesitará una copia del formulario de registro de su entidad financiera, por tanto, las CAs, deberán almacenar el formulario correspondiente a cada entidad o conocer dónde se encuentra disponible.

Los pasos a realizar serán los siguientes (figura 3):

El SW titular inicia el proceso enviando una solicitud de inicio al la CA.

La CA devuelve una respuesta firmada junto con su certificado, que incluye sus claves públicas.

El SW titular recibe la respuesta, verifica la autenticidad de la CA y genera un mensaje

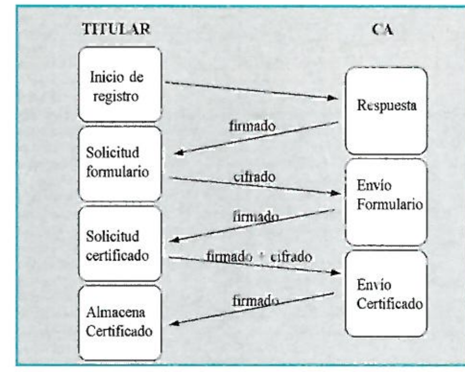


FIGURA 3. REGISTRO DEL TITULAR.

cifrado de solicitud de formulario de registro en el que incluye su número de cuenta.

La CA deduce cual es el formulario de registro correspondienta a la entidad relacionada con el titular a partir de su número de cuenta y lo devuelve firmado.

El titular cumplimenta el formulario de registro y el software genera un mensaje de solicitud de registro con la información del formulario, cifrado y firmado.

La CA despues de verificar la aptitud del titular, crea el certificado y lo envía firmado, al mismo.

El SW titular almacena el certificado.

En esta relación se ha descrito únicamente el flujo de mensajes entre el software titular y la autoridad de certificación, sin embargo, el proceso completo consta de 32 pasos. Estos incluyen desde la verificación de los certificados hasta la generación de las firmas y de múltiples cifrados con claves de sesión. Por ejemplo, en el punto 3, el software titular tras proceder a los pasos necesarios para la verificación del mensaje recibido, debería seguir los siguientes: 1. Generación de una clave simétrica, K; 2. Encriptación con K del mensaje, M, de solicitud; 3. Cifrado de su número de cuenta, n, y la propia clave simétrica con la clave pública de la CA, Kpca[K,n]; y 4. Envío de todo ello.

REGISTRO DEL COMERCIO

Los comercios deben registrase con una CA como paso previo al proceso de transacciones SET con una pasarela de pago y para permitirles la recepción de mensajes SET de titulares. Al igual que ocurría en el proceso de registro de titulares, la comunicación con la CA se hace segura a través de la clave pública de esta última, que se extrae de su certificado digital.

El comercio también necesita un formulario de registro de su institución financiera adquiriente, que deberá identificar ante la CA. Por supuesto, esta entidad debe aceptar instrucciones de pago SET y ser capaz de realizar transacciones SET.

Los pasos a realizar serán los siguientes (figura 4) :

El SW del comercio inicia el proceso enviando una solicitud de inicio al la CA.

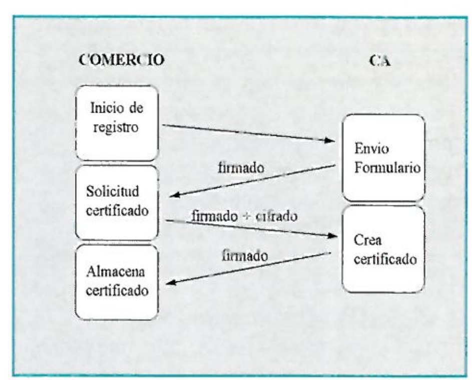


FIGURA 4. REGISTRO DEL COMERCIO.

La CA devuelve una respuesta firmada que contiene el formulario de registro junto con su certificado, que incluye sus claves públicas.

El SW del comercio recibe la respuesta y verifica la autenticidad de la CA. El comerciante cumplimenta un formulario en pantalla con información sobre el comercio y el software genera los dos pares de claves asímetricas, el destinado al intercambio de claves simétricas y el destinado a firmas digitales. Finalmente se envía todo ello en un mensaje de solicitud de certificado firmado y cifrado.

La CA genera los certificados de ambas claves y los envía con un mensaje firmado al comercio, que los almacena para su posterior uso.

PROCESO DE COMPRA

El proceso básico de compra, que se describe a continuación, se puede observar en la figura 5.

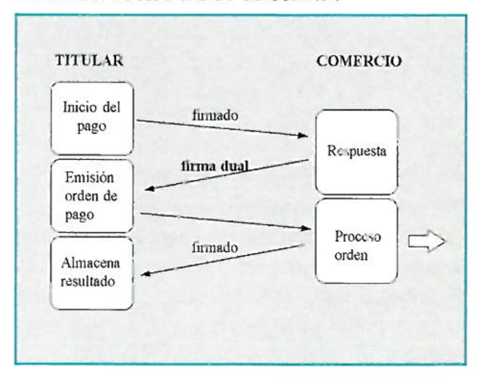
El titular necesita conocer la clave pública de la pasarela de pago con el fin de poder enviar mensajes al comercio, por ello, el primer paso consiste en solicitar una copia del certificado de la pasarela al comercio. En este primer mensaje, el titular indica ya que marca de tarjeta empleará para la transacción.

El SW del titular inicia el proceso enviando una solicitud de inicio al comercio.

El comercio responde con un mensaje firmado en el que incluye el certificado de intercambio de la pasarela y el suyo propio de firma.

El software del titular genera la información de pedido (OI) y las instrucciones de pago (PI) y

FIGURA 5. PROCESO BASICO DE COMPRA.





Otras transacciones set

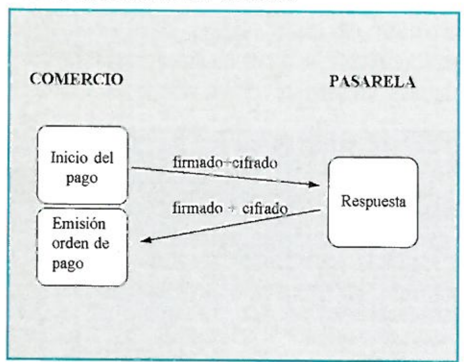
Además del flujo "habitual" de las transacciones SET, el protocolo contempla también otras transacciones y mensajes:
Solititud de estado sobre certificado: **En ocasiones, una CA no puede completar rápidamente el proceso de una solicitud de certificación. En estos casos envía un mensaje al solicitante indicándole que pruebe más tarde. El solicitante podrá entonces enviar un mensaje para pedir información sobre el avance del proceso de la CA para recibir información del estado del mismo o el propio certificado si el proceso ha terminado.**
Solicitud de estado de compra: **Permite que el software del titular pregunte al del comerciante sobre el estado de su compra una vez que ha recibido la aceptación procedente del mismo. Este es el paso que se ha indicado como 7 en la figura 2.**
Anulación de pedido: **Permite la anulación completa de un pedido completo o parte del mismo.**
Anulación de captura de datos: **Permite que un comerciante corrija errores en la captura de datos.**
Dotación de crédito un titular: **Permite que un comerciente dote de crédito la cuenta de un titular. Esto es necesario, por ejemplo, cuando se devuelve la compra o resulta dañada durante el envío. Este mensaje, siempre es generado por el comerciante, no por el titular. Todas las comunicaciones que se llevan a cabo entre titular y comerciante y que dan lugar a la devoluvión de crédito, escapan al ámbito de la especificación y, por tanto, se llevan a cabo fuera de SET.**
Anulación de crédito: **Permite que un comerciante corrija una mensaje de crédito previo.**
Administración batch: **Consiste en la comunicación de información entre el comerciante y la pasarela de pago respecto a lotes de comerciante.**
Error: **Indica que un mensaje es rechazado por formato inconveniente o por fallo en la verificación de su integridad.**

coloca en ambas un identificador de transacción asignado por el comercio. Este identificador será el que permitirá a la pasarela de pago relacionar OI y PI entre sí cuando el comerciante solicite autorización. Tras ello, genera una firma dual de OI y PI y firma ambas. Además cifra PI para la pasarela de pago y envía el conjunto junto a las huellas digitales de OI y PI al comercio.
El comerciante verifica el mensaje y ejecuta el proceso de autorización de pago (descrito en el punto siguiente). Tras ello, genera un mensaje de confirmación y lo devuelve firmado al titular. Como se indicaba anteriormente, el orden de ejecución de los pasos en este punto puede variar. En algunos casos se ejecutará el proceso de autorización de pago con posterioridad al envío del mensaje de confirmación al titular.

AUTORIZACION DE PAGO

Una vez un comercio ha recibido una orden de compra, la debe procesar a través de la pasarela de pago (figura 6).
El comercio genera y firma digitalmente una solicitud de autorización que incluye la cantidad a ser autorizada y el identificador de la transacción y la envía a la pasarela de pago cifrada con la clave pública de la misma junto con las instrucciones de pago que recibió del titular ya encriptadas con esta misma clave.
La pasarela de pago verifica la integridad del mensaje del comercio y su autenticidad a través del certificado del mismo y verifica la integridad, autenticidad y correspondencia de las instrucciones de pago con la orden de pago (cuya huella digital proviene de la firma dual).
Si todo es correcto, genera un mensaje de solicitud de autorización a través de la red privada del sistema de pago (A).
Tras la recepción de una respuesta positiva (B) del emisor de la tarjeta, la pasarela genera, firma digitalmente y cifra un mensaje de respuesta de autorización que devuelve al comercio.
El comerciante verifica el mensaje y, en caso de tratarse de una aceptación completa el proceso de la orden de compra distribuyendo los bienes o llevando a cabo los servicios solicitados.

FIGURA 6. AUTORIZACION DE PAGO.



EJECUCION DEL PAGO

La autorización de pago no implica que se lleve a cabo la transferencia entre el banco emisor y el banco adquiriente. Para que ésta se haga efectiva, el comercio debe emitir un mensaje de ejecución de pago a la pasarela. El flujo de comunicación es totalmente análogo al de la autorización de pago y la única nota a destacar es que el protocolo permite que en un único mensaje se realice la solicitud de pago de un lote de transacciones diferente. Este es el paso que se ha indicado como 8 en la figura 2.

HOMOLOGACION DE SOFTWARE SET

Para que realmente pueda convertirse en un estándar, SET debe permitir el rápido desarrollo de aplicaciones comerciales. Así pues, desde el principio, uno de los objetivos básicos es el de lograr interoperabilidad entre aplicaciones desarrolladas por distintos fabricantes, soportar todos los estándares abiertos de pago, basarse en los estándares ya establecidos siempre que sea posible y conseguir un sistema no restringido por leyes de exportación. Por ello, además de la definición de una especificación, es necesario acompañarla de un proceso de homologación. Hasta el momento se ha realizado una implementación de referencia y, en este momento, se sigue aún diseñando una estructura de tests de interoperabilidad.

a) Pruebas de compatibilidad
El primer paso que debe seguir una implementación software para acceder a una homologación SET es la verificación de la capacidad de interactuar correctamente con la implementación del borrador de la referencia (DRI). SAIC/Tenth Mountain Systems ha generado un conjunt de 50 mensajes de prueba que se encuentran dispobibles para todos los fabricantes en su servidor (http://www.tenthmountain.com/html/compatibility.html). Existe una versión del DRI compilable a un ejecutable que se ha distribuido ya a más de 300 solicitantes.

El objetivo de SET no es proporcionar seguridad a los pedidos, sino a los medios de pago

Una vez que el fabricante ha terminado de procesar los 50 mensajes de prueba, debe enviar el resultado al servidor de TMS que determinará si se ha pasado la prueba. En caso contrario, el personal de TMS informará de cuales han sifo los fallos, cuáles podrían ser las causas y cuáles las posibles soluciones. Una vez las pruebas han sido superadas, se le asigna el status de "fabricante SET" al fabricante del software y se le incorpora a las listas de fabricantes compatibles.

b) Pruebas de Interoperabilidad
Además de superar las pruebas de compatibilidad con la implementación de referencia, se requiere que las aplicaciones componentes SET de los distintos fabricantes sean capaces de interoperar entre sí. Por tanto, se realizan una serie de pruebas que deben involucrar no sólo los propios mensajes SET, sino tambien las porciones de aplicación



Bibliografía y referencias

Las referencias aluden a artículos del autor en Programación Actual correspondientes a los números:
[Echeva-1] "Medios de pago no efectivo electrónicos", núm 10.
[Echeva-2] "Canales seguros y autenticación", núm. 11
[Echeva-3] "Conceptos básicos de seguridad", núm. 7
• La bibliografía básica sobre SET es la propia especificación, que se ha incluido en el CD-ROM.

Bibliografía complementaria es:
• "David Chaum on electronic commerce", IEEE Internet Computing, Vol I, num. 6.
• IEEE Spectrum, Special Issue on Electronic Commerce, Feb. 97
Computer
• N. Asokan et Al. "The State of the Art in Electronic Payment Systems", IEEE Computer, Sept 1997

Además es posible encontrar información en:
http://www.visa.com
http://www.mastercard.com

dependientes del fabricante que quedan fuera del ámbito de lo exigido por SET.
En este momento, aún no se encuentra definido un test de interoperabilidad ya que éste debe ser lo más cercano al modo en que el software interoperaría en un caso real, por lo que las pruebas en batch no son apropiadas. Se está buscando un proceso que permita a las asociaciones arbitrar cuando se producen discrepancias en el modo en que dos fabricantes consideran que se debe implementar un determinado aspecto de la especificación. Por el momento, son los propios fabricantes los que realizan pruebas entre sí y contabilizan los casos en los que han resultado ser compatibles con el software de otros fabricantes.

El certificado del comprador equivale a la firma del dorso de una tarjeta de crédito

c) Pruebas de homologación
Tras la finalización de las pruebas anteriores, los fabricantes deben solicitar la certificación de sus productos como homologados SET. Para ello, deberán entrar en un proceso de certificación controlado por una tercera parte neutral y pagar una cuota por cada componente software que desean les sea certificado (monedero, servicor del comerciante, pasarela de pago o autoridad de certificación).
d) Experiencias piloto
Hasta el momento se han llevado a cabo diversas experiencias piloto mixtas con SET: Chase y Wal-Mart, Mellon Bank y el Deartamento del Tesoro de los Estados Unidos, Credit Union Electronic Transaction Services (CUETS) y SaskTel, Commerzbank y Karstadt, ChinaTrust Commercial Bank y Citibank, UC Card Japan y UC Cybermall, Amalgamated Banks of South Africa (ABSA) y South African Certification Agency (Pty) Ltd. y Danish Payment Systems (PBS). Sin embargo, aún no se ha definido una política de homologación y pruebas de interoperabilidad.
La última lista oficial de comerciantes es del 14 de Abril de 1997.

CONCLUSIONES
Hay que reconocer el aspecto positivo de SET en el sentido de se trata de un estándar desarrollado de manera abierta, involucrando a multitud de fabricantes e instituciones financieras, con los comentarios y revisiones del "público".
Si bien es cierto que SET es un sistema que apuntalaría la actual infraestructura de pago a través de tarjetas de crédito en un entorno electrónico, no presenta ningún elemento que suponga una reducción de costes sino, más bien, todo lo contrario. Los bancos emisores de tarjetas deberán afrontar el coste económico que supone el reparto de pares de claves a los usuarios de sus tarjetas, y lo que es más importante, la certificación de las mismas por una autoridad de certificación, coste que, muy probablemente, será soportado por los titulares o los comerciantes asociados. De momento, este tema se ha dejado en suspenso.
Por otra parte, los beneficios que supone para las entidades emisoras, aún no están muy claros. La reducción en el precio del software que vendrá dada por la utilización de un estándar, beneficiará, en todo caso, a consumidores y comerciantes y la disminución del fraude será una ventaja para las entidades adquirientes, no para las emisoras, pues son las primeras las que asumen el riesgo ante un fraude. Por otra parte, no está claro si SET generará un mayor uso de las tarjetas de crédito o se limitará a desplazar el uso de las compras por correo y por teléfono a las redes telemáticas sin generar nuevo mercado. En este caso, SET podría disminuir los beneficios de los emisores de las tarjetas, lo que supondría una recalificación de las comisiones establecidas.
Respecto a la internacionalización del software en función de las restricciones a la exportación, RSA Data Security ha obtenido licencia para la exportación en software SET a todo el mundo. Sin embargo, aunque este obstáculo ha sido salvado, el modelo de negocio no se adapta a los de todos los países. Por ejemplo, la mayoría de los pagos electrónicos en Japón se realizan mediante transferencia directa entre cuentas, no a través de tarjetas. De esta forma, no se pagan comisiones.

El titular debe tener, al menos, tantos certificados como tarjetas de crédito

Por otra parte, como bien señala Chaum, el secreto mejor guardado del comercio electrónico es que la técnica cuestiona ya toda la infraestructura clásica de los cobros a través de tarjeta que SET trata de mantener. Simplemente, no existe ya necesidad de un banco adquiriente, ya que este rol puede ser desempeñado por el propio comercio.
La implantación del sistema se está retrasando más de lo previsto, inicialmente se había anunciado para el primer trimestre de 1997 y sufrió un retraso al plantearse ese año como dedicado a pruebas de laboratorio y de campo. Más tarde se anunción su puesta en explotación para finales del 97 y, a comienzos del 98 aún sigue sin definirse el procedimiento de homologación y la lista oficial de fabricantes no ha sido renovada desde mediados del año pasado.
Como señalan algunos analistas no parece que el estándar SET lo vaya a ser de facto antes del 99 y quizá antes del próximo milenio, aunque sí parece que antes, o después, lo será, no por sus bondades, sino por quien se encuentra detrás y los intereses que defiende.

Listas de correo

mailto :majordomo@commerce.net

Anuncios acerca de la especificación
Body : subscribe SET-announce
Foro de discusión y comentarios
Body: subscribe SET-discuss

Autor: Fernando de la Villa

Powersoft PowerJ 2.0 Enterprise

PowerJ 2.0 Enterprise es un completo entorno de desarrollo de programas Java. Permite la creación tanto de pequeños applets como de grandes aplicaciones que hagan un intensivo uso de las bases de datos.

Powersoft vuelve a la carga con un completo entorno dedicado a Java, el lenguaje de moda. El uso de Java está muy extendido en la actualidad, pero en la mayoría de los casos se utiliza para la creación de pequeñas aplicaciones y de applets para mejorar un sitio web. Sin embargo, es un lenguaje capaz de mucho más. No en vano está basado en C++, un lenguaje que ha demostrado ser perfectamente válido para el desarrollo de grandes aplicaciones. PowerJ se ha creado pensando en los desarrolladores que buscan sacar el máximo partido a Java.

Entre sus características destaca el amplio soporte a la creación de aplicaciones que utilizan bases de datos. Incluye diversos componentes, herramientas y manejadores especializados en este aspecto. Proporciona soporte para JavaBeans, ActiveX y CORBA. Además, ofrece cientos de componentes y clases Java, incluye una extensa documentación en línea y multitud de programas de ejemplo.

COMPLEMENTOS

El producto viene acompañado por un buen número de elementos que le proporcionan gran parte de su potencia de desarrollo de aplicaciones Java. A continuación se realiza una breve descripción de los complementos incluidos en el producto:

1) Componentes Java.

• ObjectSpace JGL: es un conjunto de clases procedente de diversas colecciones. Incluye soporte para listas y multitud de útiles algoritmos destinados a ahorrar trabajo al programador. Existen dos versiones, una para el kit de desarrollo de Java 1.02 y otra para el 1.1.

• JScape Widgets y Widgets Pro: se trata de una colección de 15 componentes, entre los que destacan la lista multicolumna, el indicador de progreso, el control de rejilla y el menú de contexto. Incluso incluye un componente para utilizar calendarios.

• Jscape Lite: son versiones limitadas de otros componentes de Jscape (PowerPanel, PowerSearch, Mail Widgets y Animations). Sólo se pueden utilizar con proyectos que utilicen el JDK 1.1.

• Jclass BWT, Chart y LiveTable: son tres componentes de KL Group. El primero extiende las capacidades del paquete AWT de Java con cerca de 20 elementos, mientras que el segundo proporciona soporte para gráficas. El último de ellos es una rejilla para la gestión de datos en tablas.

2) Complementos orientados a las bases de datos.

• Sybase SQL Anywhere: el conocido sistema gestor de bases de datos de Sybase. Permite la creación de aplicaciones cliente/servidor gracias a sus capacidades de gestión de transacciones. Sybase SQL Anywhere proporciona el acceso a un buen número de formatos de base de datos y ofrece un completo soporte para el uso del lenguaje SQL en las aplicaciones.

• Sybase jConnect para JDBC: es la versión de desarrollo del manejador JDBC de Sybase. Mediante su uso se consigue un acceso nativo a otros productos de la compañía, como SQL Server y SQL Anywhere.

• Sybase NetImpact Dynamo: herramientas destinadas a la creación y gestión de webs que proporcionan acceso a información almacenada en bases de datos.

• Sybase SQL Tools: incluye una serie de utilidades que facilitan la utilización del lenguaje SQL en los proyectos desarrollados con PowerJ. Destacan Sybase Central, una herramienta de administración de bases de datos, y Open Server Gateway, elemento imprescindible para poder utilizar conjuntamente jConnect y SQL Anywhere.

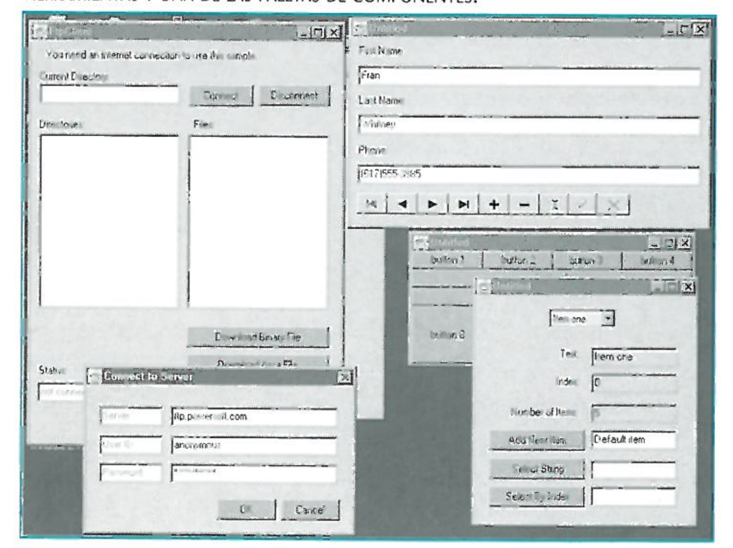
Instalación

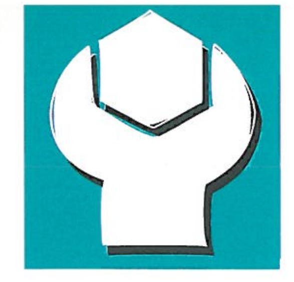
PowerJ es una herramienta de desarrollo un tanto exigente en cuanto a los requerimientos mínimos se refiere.

Para instalar y ejecutar correctamente el producto se necesita una máquina que cumpla los siguientes requisitos:

• Procesador: Pentium 75 MHz o superior.
• Memoria: 32 megas de RAM.
• Sistema operativo: Windows 95 o Windows NT 4.0 (o superiores).
• Unidad lectora de CD-ROM.
• Espacio libre en disco duro: se necesita una media de 230 megas. El tamaño total depende, como suele ser habitual, de los elementos que se desean instalar y del tamaño de cluster del disco duro destino. Por ejemplo, el soporte de la versión 1.0.2 del kit de desarrollo de Java (JDK) necesita 110 megas libres, mientras que el soporte para la versión 1.1 del JDK requiere 150 megas. Como no existe la posibilidad de ejecutar desde el CD-ROM, lo mejor es instalar todo el producto (más de 600 megas) para no tener ningún tipo de problemas.

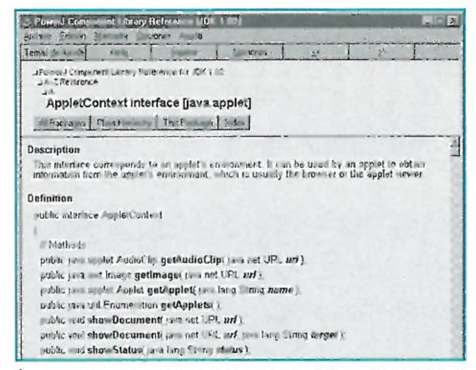
DETALLE DEL ENTORNO DE DESARROLLO EN EL QUE SE PUEDE VER LA BARRA DE HERRAMIENTAS Y UNA DE LAS PALETAS DE COMPONENTES.



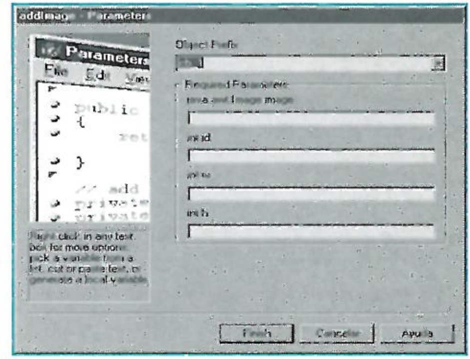




La tarjeta de referencia es una herramienta sumamente util para los programadores menos experimentados.



La tarjeta de referencia proporciona un facil y rapido acceso a la ayuda en linea del lenguaje de programacion.



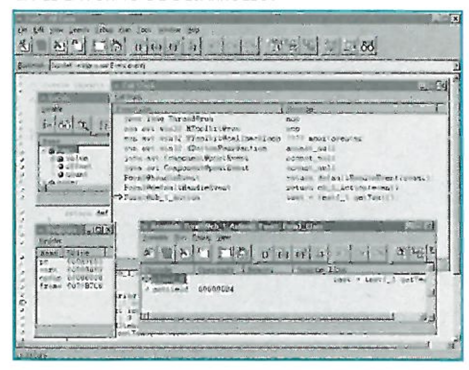
El asistente de parametros proporciona una manera muy sencilla de rellenar los parametros necesarios para realizar la llamada a una funcion.

• XDB JetConnect: es un manejador JDBC que permite el acceso a bases de datos mediante ODBC y JDBC. Por supuesto, se trata de la edición de desarrollo.
• Visigenic VisiChannel para JDBC: similar al anterior, es un manejador JDBC que utiliza el protocolo IIOP.
3) Otros complementos.
• Powersoft ObjectCycle: es un producto orientado al control de versiones. Resulta especialmente útil en la creación de grandes aplicaciones.
• Powersoft Jaguar CTS: la edición de PowerJ objeto del presente artículo contiene una versión beta de Jaguar CTS para NT. Es un servidor de transacciones de componentes diseñado para el desarrollo de aplicaciones transaccionales escalables para NetOLTP. Soporta diferentes modelos de componentes y proporciona gestión de conexiones y de sesiones, además de conectividad con múltiples bases de datos.
• Sun Java Development Kit (JDK): no podía faltar el kit de desarrollo Java de Sun Microsystems. PowerJ incluye las versiones 1.02 y 1.1.
• Thinking in Java: el paquete viene acompañado por la edición electrónica del libro "Thinking in Java", escrito por Bruce Eckel. Para leerlo es necesario el programa Adobe Acrobat Reader, incluido en el CD-ROM de instalación.

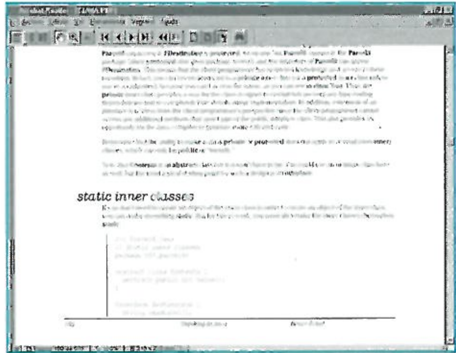
EL ENTORNO

PowerJ ofrece al programador un entorno de trabajo intuitivo y sencillo. Sus características son similares a los entornos integrados de otros productos de la compañía, por lo que los usuarios con experiencia no tendrán ningún problema en dominarlo. De todos modos se trata de un entorno muy fácil de aprender. Está formado por los siguientes elementos principales:
• Barra de herramientas: proporciona un acceso directo a las opciones más utilizadas del entorno, como abrir un proyecto, almacenar un formulario o ejecutar la aplicación.
• Paletas de componentes: se trata de una colección de componentes susceptibles de ser utilizados en cualquier aplicación Java. Están ordenados por categorías seleccionables mediante distintos separadores. Existen dos paletas principales. La primera está dedicada a los componentes de la versión 1.02 de Java, mientras que la segunda contiene los de la versión 1.1. PowerJ muestra de forma automática las paletas apropiadas según las características de cada proyecto.
• Inspector de objetos: sirve para revisar o modificar las propiedades y los eventos relacionados con cada tipo de objeto. El Inspector muestra todas las propiedades de un objeto, incluso aquellas de uso interno que no pueden ser modificadas por el usuario.
• Ventana de objetos: contiene una lista de los objetos que forman la aplicación, ordenados por el formulario al que pertenecen. La lista se muestra de forma jerárquica, con los objetivos del proyecto (targets) en la raíz. Esta ventana proporciona un método alternativo para modificar el contenido de los formularios.
• Ventana de clases: muestra una lista con todas las clases utilizadas en la aplicación. Cuando se selecciona cualquiera de ellas se obtiene el acceso a los eventos, las propiedades y las funciones miembro. Se pueden añadir y eliminar elementos a la clase, y se puede editar la porción de programa relacionada con ellos.
• Ventana de archivos: como su propio nombre indica, se trata de una ventana dedicada a los archivos que componen la aplicación. Permite la adición de nuevos archivos, así como la eliminación y edición de archivos existentes.
• Editor de código: es un potente editor de código fuente ampliamente utilizado en el desarrollo de aplicaciones con PowerJ. Entre sus características destacan el soporte de la programación mediante arrastrar y soltar, coloreado de la sintaxis, indentado automático y control de depuración.

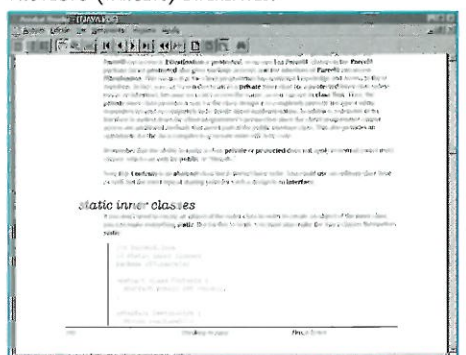
PowerJ incluye un potente depurador integrado en el entorno de desarrollo.

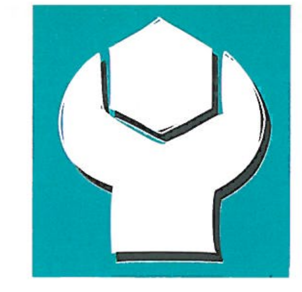


Las vistas del depurador permiten examinar hasta el ultimo detalle del programa en ejecucion.



PowerJ admite hasta un total de 14 objetivos de proyecto (targets) diferentes.

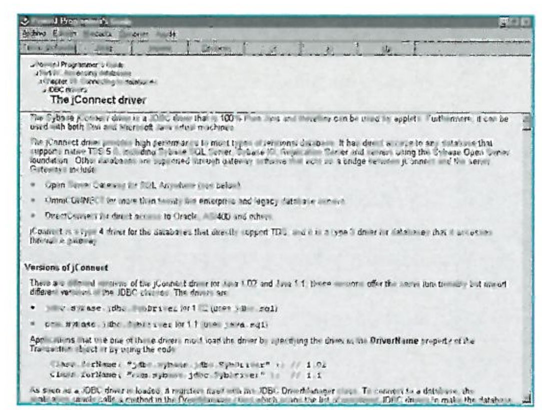




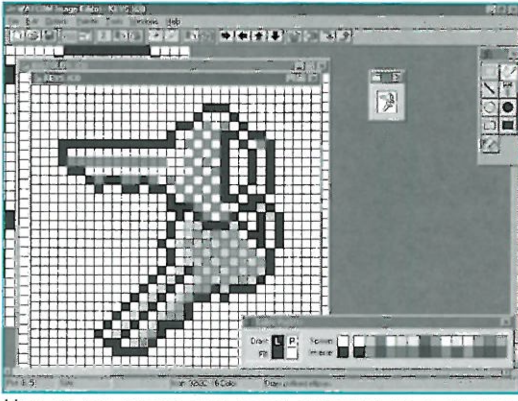
• Tarjeta de referencia: se trata de una herramienta sumamente útil para el programador. Muestra una estructura jerárquica con todas las clases Java soportadas por PowerJ. Para cada clase proporciona una lista de todos los métodos y atributos que la componen. Permite la inserción de código en la ventana de edición y proporciona el acceso a la ayuda en línea asociada al elemento seleccionado.
• Asistente de parámetros: es un complemento de la tarjeta de referencia. Se utiliza para rellenar cómodamente los parámetros de las funciones.

DEPURACION DE PROGRAMAS

La depuración es una tarea esencial en la creación de programas en cualquier lenguaje de programación. Cuando se trata de lenguajes como C++ y Java, cobra una mayor importancia debido a las características de la programación orientada a objetos. De hecho, un entorno de desarrollo que no disponga de opciones de depuración no tiene mucho sentido. Por fortuna, PowerJ incorpora un depurador de programas Java repleto de interesantes características.
El depurador proporciona al programador una completa gestión de puntos de ruptura. El funcionamiento habitual de estos puntos consiste en detener la ejecución para que se pueda examinar el estado del programa en ese momento. PowerJ, además, permite puntos de ruptura condicionales que se activan cuando se cumple una condición o cuando se ha pasado por ellos un determinado número de veces. También es posible ejecutar una sentencia Java cuando se activan.



EN LA DISTRIBUCION ESTA INCLUIDA LA EDICION ELECTRONICA DEL LIBRO "THINKING IN JAVA" DE BRUCE ECKEL.



UNO DE LOS PUNTOS FUERTES DEL PRODUCTO ES LA EXTENSA AYUDA EN LINEA DE LA QUE DISPONE.

Por comodidad del programador se ha contemplado la opción de no parar la ejecución al llegar a uno de estos puntos, pero sí ejecutar una sentencia Java.

PowerJ ofrece al programador un entorno de trabajo intuitivo y sencillo

El programa en ejecución puede ser estudiado desde diferentes puntos de vista. Para ello existen distintas ventanas especializadas, cada una de ellas, en un aspecto concreto:
• Call Stack: proporciona la secuencia de llamadas a función que se ha producido desde el comienzo del programa hasta el punto de ruptura. La ventana muestra una lista con el nombre de las funciones. Cuando se pulsa en cualquiera de ellos, el entorno de depuración cambia para mostrar información sobre la nueva función seleccionada.
• Locals: permite el estudio de los valores contenidos en las variables locales definidas en la función que estaba en ejecución. Muestra una lista con el nombre de la variables, sus valores actuales y el tipo de datos al que pertenecen.
Además, utiliza una serie de iconos para identificar con facilidad el tipo de la variable. Por ejemplo, los punteros se destacan con una pequeña flecha, las variables de tipo elemental con una esfera de color amarillo y los objetos con cajas de diferentes colores. Desde esta ventana se permite cambiar directamente el valor de las variables y se puede examinar el contenido de la memoria asociada.
• Watches: esta ventana se utiliza para realizar un seguimiento de los valores que toma una variable a lo largo de la ejecución de la aplicación. Es más, no sólo es posible estudiar una variable, sino cualquier expresión válida en Java. Esta característica resulta especialmente útil para acceder a los valores de las variables contenidas en la clase padre, cosa que no es factible desde la ventana Locals.
• Assembly: sirve para examinar el código ensamblador de la aplicación. Cada línea mostrada en la ventana contiene el código de operación, los operandos, la referencia a memoria y la línea de código fuente. Además, es posible ver la dirección de memoria donde está almacenada la instrucción. El único inconveniente de la ventana Assembly reside en que sólo es útil si se utiliza la máquina virtual Java de Microsoft, en lugar de la máquina virtual de Sun.
• Registers: la misión de esta ventana es mostrar los contenidos de los registros hardware de la máquina. También permite cambiar el valor de los mismos.
• Threads: especialmente útil para depurar los hilos de ejecución. Proporciona el identificador de cada hilo y su estado actual.

POWERJ INCLUYE UNA UTILIDAD EXTERNA PARA LA EDICION DE ARCHIVOS GRAFICOS.

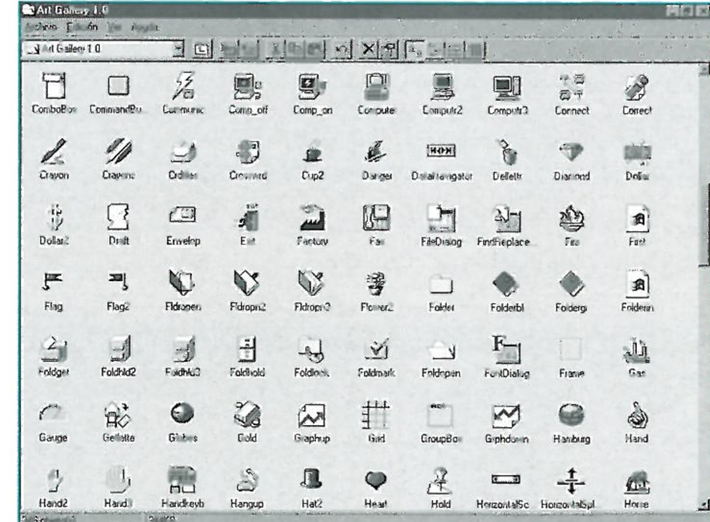
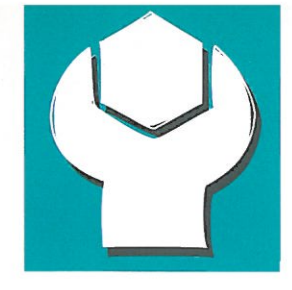
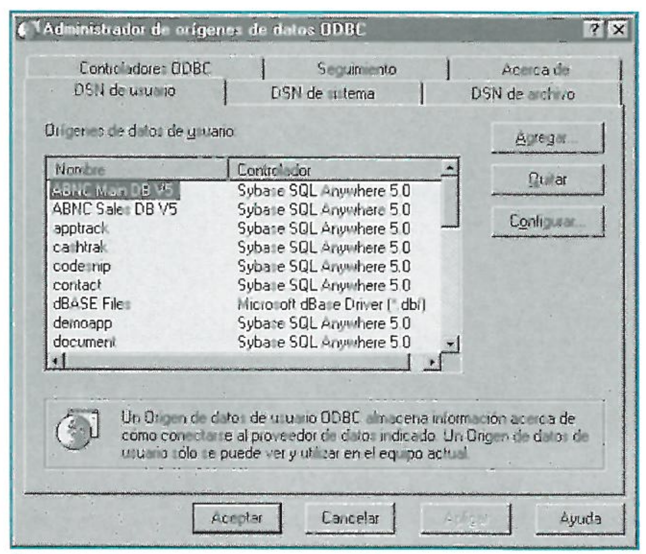


TABLA 1. REQUERIMIENTOS DEL SISTEMA

Procesador	Pentium 75 MHz o superior
Memoria RAM	32 megas
Disco duro	230 megas (en media)
Sistema Operativo	Windows 95 o Windows NT 4.0 (o superiores)
Otros	Unidad lectora de CR-ROM





El programador dispone de una amplia y variada coleccion de iconos para utilizar en sus aplicaciones.

• Memory: se utiliza para examinar el contenido de una zona de memoria, aunque en la realidad sólo sirve para estudiar el contenido de un array.

PowerJ incluye una clase denominada "Debug" que posibilita la depuración de una aplicación añadiendo determinadas líneas de código al programa existente. Proporciona aserciones y excepciones, además de una serie de funciones específicas para mostrar información y utilizar un registro de depuración.

DESARROLLO CON POWERJ

El desarrollo de aplicaciones con PowerJ gira en torno al concepto de proyecto. Los proyectos aglutinan todos los archivos de una aplicación y contienen información acerca del modo de construcción del ejecutable. En cuanto a ejecutables, se pueden crear programas y applets Java, clases, archivos CAB y JAR, y servidores para la web. Se pueden especificar nada más y nada menos que 14 objetivos diferentes para un proyecto. Como ya se ha dicho, PowerJ no es producto destinado al aficionado que desea añadir applets a su página web. Es un completo entorno para desarrollar grandes programas Java, y por ello muchas de sus características están orientadas a las bases de datos. Es posible crear tanto aplicaciones Java como servidores para la web que proporcionan acceso a la información contenida en una base de datos.

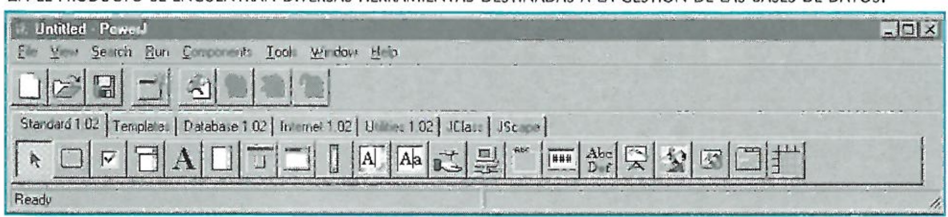
La mecánica de creación de una aplicación es similar a la de la mayoría de herramientas visuales de desarrollo disponibles en el mercado. En primer lugar se crea la interfaz de usuario, a continuación se especifican las propiedades de los objetos creados y, por último, se añade el código que responde a los eventos. Estos tres pasos se repiten una y otra vez para las diferentes partes de la aplicación hasta obtener el resultado final.

Exceptuando los menús de la aplicación, la interfaz de usuario gira en torno a los formularios. Sobre ellos se sitúan los diversos componentes (cajas de texto, listas, imágenes, etcétera) que permitirán al usuario interactuar con la aplicación. Para crear un nuevo formulario basta con pulsar el botón de la barra de herramientas destinado a tal efecto.

Es una herramienta de desarrollo un tanto exigente con respecto a los requerimientos mínimos del sistema

El siguiente paso consiste en seleccionar componentes de la paleta y situarlos sobre el formulario. PowerJ crea cada uno de ellos con un tamaño estándar, pero es posible crearlos directamente con un tamaño determinado mediante el procedimiento de arrastrar y soltar. Una vez colocados todos los componentes se puede mejorar la estética del formulario. Esto se consigue con la ayuda de las opciones de alineado y ajuste de tamaño incorporadas en el entorno. Cuando se consigue el resultado deseado sólo queda añadir el código necesario en la ventana de edición. Para ello el usuario cuenta con la valiosa ayuda de la tarjeta de referencia antes mencionada y de la programación mediante arrastrar y soltar.

En el producto se encuentran diversas herramientas destinadas a la gestion de las bases de datos.



PROGRAMACION MEDIANTE ARRASTRAR Y SOLTAR

El lenguaje de programación Java dispone de una amplia jerarquía de clases. Todas ellas están repletas de métodos y atributos difíciles de memorizar. En el caso de los métodos el asunto se agrava notablemente debido al elevado número de parámetros que sería necesario aprenderse. Para aliviar el problema, el entorno de PowerJ incorpora la posibilidad de programar mediante la técnica de arrastrar y soltar. Se trata de un sistema ideal para principiantes y usuarios ocasionales que facilita enormemente la tarea de editar código.

A la hora de escribir el programa fuente se dispone de dos opciones. La primera es teclear todo el código de la aplicación en la ventana de edición. La segunda consiste en combinar la anterior con la programación mediante arrastrar y soltar. Cuando se va a hacer referencia a un componente, este se arrastra desde el formulario hasta la línea de código deseada en la ventana de edición. En ese momento se abre de forma automática la tarjeta de referencia, en la que aparece seleccionada directamente la jerarquía de la clase a la que pertenece el componente. Acto seguido se busca el método al que se desea hacer referencia y se pulsa sobre el botón apropiado para invocar al asistente de parámetros. Por último, se rellenan los valores necesarios para que se inserte en el programa la línea de código correspondiente.

Sin duda, es un sistema válido para programadores con poca experiencia, que acuden a la ayuda en línea frecuentemente. Pero es bastante incómodo para un programador experimentado, que siempre irá más rápido tecleando directamente el código. Aún así, en determinadas ocasiones resulta inevitable acudir a este sistema porque se ha olvidado el nombre de uno de los métodos o el tipo de algún parámetro.

Consideraciones finales

Sin duda, se trata de un producto muy completo e interesante. Es fácil de usar y permite la creación de un amplio abanico de proyectos en Java. Además dispone de infinidad de complementos y una abundante documentación en línea que le convierten en entorno de desarrollo realmente profesional.

Pero, ni por su precio ni por sus características, está dirigido al usuario ocasional de Java. Sin embargo, puede ser la solución ideal para las empresas que desean trabajar en serio con este lenguaje.

CONTENIDO DEL CD-ROM

Software GNU

A continuación detallamos el contenido del CD-ROM del número 13 de PA.

En el cd-rom de este mes se encuentra además de los fuentes y utilidades relacionadas con los distintos cursos, gran cantidad de software gnu, utilidades share de programación, los últimos navegadores para la red y las demos operativas de DIV GAMES STUDIO y VISUAL CAFE de Symantec.

DEMOS OPERATIVAS

En el directorio \Symantec se encuentran las demos de los siguientes productos de Symantec
Norton Antivirus para Windows 95, NT 4.0 y 3.51.
Norton CrashGuard.
Visual Cafe 1.0.
En el directorio \DIV la demo operativa de DIV GAMES STUDIO, toda la información en el manual de regalo.

GARGOSCENE

En \GARGO la revista electrónica Gargoscene desarrollada por miembros de la asociación de desarrolladores STRATOS. En esta ocasión el número 1. Si os gusta

SOFTWARE GNU

En \GNU el siguiente software distribuible bajo licencia GNU:

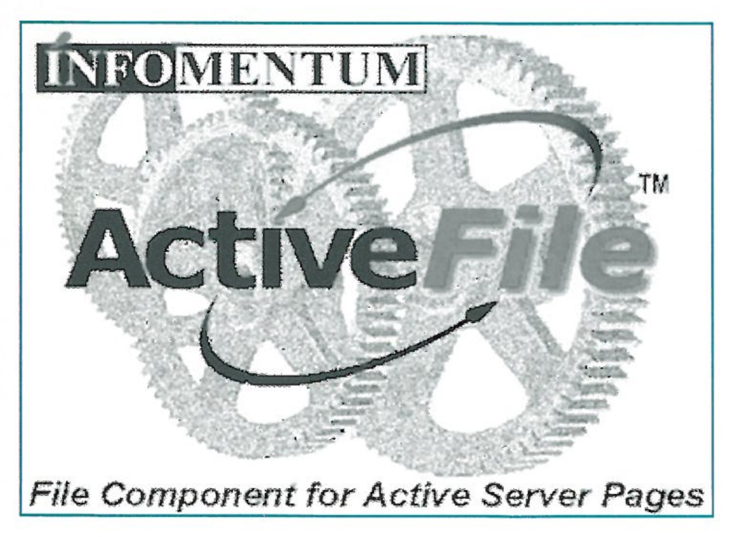
Compilador GNU C++ v2.7.2.1 para DOS

Intrucciones de instalación: Para trabajar con el compilador deberá crear un directorio llamado CC en su disco duro, a partir del cual se descomprimiran los ficheros necesarios. La instalación mínima serán los ficheros que se encuentran en el directorio \DJGPP\V2, junto con los ficheros que comiencen por GCC272 del directorio \DJGPP\V2APPS, los cuales deberán ser descomprimidos en el directorio \CC de su disco duro.
La forma de proceder podría ser la siguiente si la unidad de su CD-ROM fuera la D:

```
C:
CD\
MD CC
CD CC
PKUNZIP -D -O
D:\DJGPP\V2\*.ZIP
PKUNZIP -D -O
D:\DJGPP\V2APPS\GCC272*.ZIP
```

Para terminar de configurar el entorno tendrá que añadir o sustituir las siguientes líneas en el fichero de configuración CONFIG.SYS:

```
FILES=15
SHELL C:\COMMAND.COM C:\
/e:1000 /p
```



Los valores 15 y 1000 son los mínimos recomendados. Si se compilan proyectos muy grandes puede ser necesario aumentar dichos parámetros. Después de esto, en el fichero AUTOEXEC.BAT hay que hacer los siguientes cambios:
En la línea PATH hay que añadir al final: ";C:\CC\BIN"
Hay que añadir una línea: "CALL SETDJGPP C:\CC C:/CC"
La instalación que se hace es básica, en el directorio DJGPP del CD-ROM se encuentran diferentes utilidades que se distribuyen con el compilador.
Para descomprimir los ficheros que les indicamos es conveniente utilizar el programa WINZIP que incluimos en el directorio \UTILS\VARIOS\WINZIP ya que existen ficheros con nombres largos de Windows 95.

Compilador GNU C++ v2.80 para Windows 95

La forma de proceder para instalar el compilador será muy parecida ya que habrá que descomprimir, siempre con WINZIP, el contenido de los ficheros que se encuentran en \GCC-WIN32 a un directorio. Incluye ficheros de ayuda para la correcta configuración.

Compilador GNU GCC v2.80 para Unix

Para instalar el compilador deberá poseer el sistema operativo Linux. Después deberá descomprimir los ficheros que se encuentran en el directorio \LINUX en su disco duro. En el fichero GCC-2.8.1.TAR.GZ se encuentra un fichero llamado INSTALL donde explica como instalar GNU C++.

En este directorio también se encuentran:
Librería estandar de C++ para GCC.
EGCS-versión experimental de GCC.
Pentium Group-GCC optimizado para pentium.

PROGRAMACION

En el directorio \UTILS\PROG recopilación de las últimas novedades de software share de programación:

ActiveFile 1.2

ActiveFile es un control ActiveX que está diseñado como un control para manejar ficheros y directorios desde servidores Web activos. Requeriere ActiveX.

ActiveResizer 1.3.0

ActiveResizer es un control configurable que permite ajustar automáticamente el tamaño de los controles de Visual Basic en un formulario. Incluye amplia documentación y ejemplos.

AddSoft 1.12

AddSoft ahorra tiempo al que lo utiliza, ya que busca shareware por toda la Red con ayuda de los buscadores más conocidos, como por ejemplo Winsite, Simtel o Tucows.

CGI Expert 3.03b

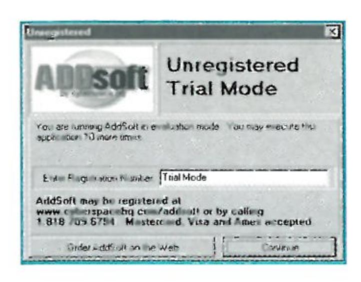
CGI, Win-CGI, ISAPI y NSAPI son interfaces que permiten crear documentos HTML dinámicos e interactivos. CGI Expert es un conjunto de componentes para Delphi 2.0 3.0 y C++ que ayudan a crear aplicaciones que soporten estos interfaces.

CGIMachine-Counter 2.1

CGIMachine-Counter es un contador CGI para páginas Web.



Contenido del CD-ROM



Incluye hasta 25 tipos diferentes de gráficos de dígitos y están incluidos en el ejecutable. La versión shareware está limitada a 200, es decir, cuando el contador llegue a 200 aparecerá un mensaje de aviso.

CodeSMART 2.2
CodeSMART es un explorador de proyectos que procesa código y crea accesos directos a rutinas, declaraciones, constantes, tipos, etc. Requiere Visual Basic 5.

CooScale 1.1
CooScale una clase de Java que permite generar una barra de progresión. En ella se pueden incluir sonidos y animaciones. Requiere entorno de desarrollo en Java con soporte para JavaBeans. No es ejecutable desde CD.

eAuthor Help 2.05 preview 5
Este poderoso programa aporta todo lo necesario para escribir ficheros de ayuda en HTML que cumplan las especificaciones estandar. Incluye extensa documentación, ejemplos y un completo tutorial.

EZ Macros 3.0
EZ Macros es un grabador de macros que soporta cientos de combinaciones de teclas "calientes" y el que se puede utilizar tanto el teclado como el ratón. Las macros pueden ser grabadas o escritas manualmente.

Form To VB Export Wizard 1.3
Form To VB Export Wizard es una utilidad para Acces 97 que permite convertir un formulario de Access a uno de Visual Basic. Como resultado se obtendría un fichero de Visual Basic 5.

HexDecCharEditor 1.02
HexDecCharEditor, como su largo nombre indica, es un editor decimal/hexadecimal que incluye búsquedas, reemplazamientos, marcas, y todo lo necesarios para hacer de



este editor una herramienta imprescindible.

Lt Directory Tree 1.2
LT Directory Tree es un control ActiveX para Visual Basic 4 y 5 que permite dar protección a unidades de disco duro o carpetas. Además da acceso directo al panel de control y permite mover directorios completos.

MultiLanguage Pack 2.01
MultiLanguage Pack es un componente para Delphi que hace que las aplicaciones puedan ser localizadas por el autor. Muy útil para crear actualizaciones o para desintalar versiones anteriores en caso de instalar una nueva.

PiXCL Tools 4.1
PiXCL Tools permite desarrollar aplicaciones gráficas aportando una gran variedad de comandos para crear y mostrar imágenes, sonidos y texto. Este lenguaje de programación interpretado ahorra el aprendizaje del complejo código del interfaz gráfico de Windows. Demo visual.

Project Analyzer 4.1.05
Project Analyzer esta diseñado para el desarrollo de programas en Visual Basic. Esta utilidad recolecta información sobre formularios, módulos, constantes y clases para generar un asistente virtual de informes para ayudar a encontrar código inutilizable.

Setup Specialist 97b
Setup Specialist es una herramienta visual para crear instalaciones de programas. Puede generarlas para plataformas de 16 y 32 bits.

Visual FoxPro 5.0 Runtime
Módulo de ejecución de Visual FoxPro 5.0, sin el cual no se podrían ejecutar los programas hechos con esta herramienta de programación.

XShell 1.0
XShell permite añadir ejecutables a los menús de Visual Basic 5. Se puede utilizar para crear accesos directos a cualquier programa desde el menú de Utilidades.

INTERNET

En el directorio \UTILS\INTERNET además de las últimas novedades share, los dos browsers más populares como son el **Internet Explorer 4.01 y Netscape Communicator 4.04.**

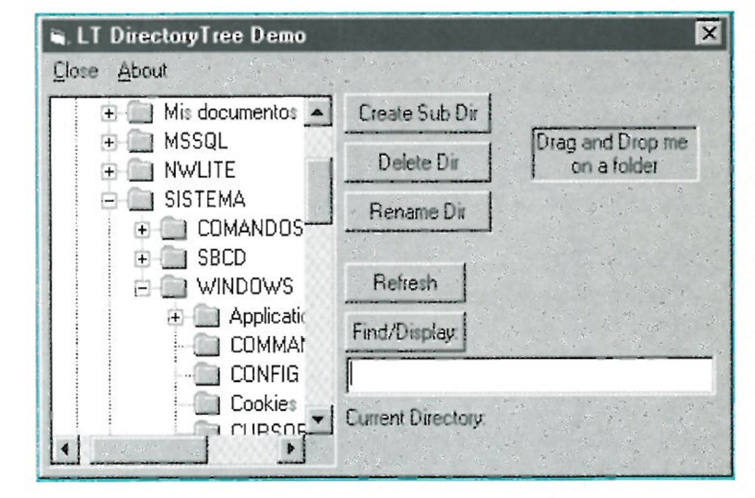
Advanced Dialer 1.2
Advanced Dialer es una pequeña utilidad para simplificar el acceso a Internet. Permite crear y mantener conexiones a la Red. Cuando se inicia, prueba varias líneas hasta que la conexión es establecida y si se corta la conexión, vuelve a marcar.

FastNet 1.0
FastNet permite acelerar la conexión Internet. Esta utilidad ahorrará tiempo a todos los usuarios de Windows 95.

Forever Connected 2.0
Forever Connected es un plug-in para Windows que controla la conexión Internet y, en el caso de que se pierda, vuelve a conectar automáticamente. Requiere las librerías: MFC42.DLL y MSVCRT.DLL.

VARIOS

En \UTILS\VARIOS recopilación de utilidades de interés general como por ejemplo: Paint Shop Pro, GWS, Acdsee, SEA, Antivirus Anyware y VirusScan, Winzip y Acrobat Reader.



DEMO 3D

En este directorio se encuentra la demo motivo del curso que sobre Entornos 3D se viene cubriendo en la revista desde el primer número.

FUENTES DE LOS ARTICULOS

En el directorio PA13 se encuentran los fuentes correspondientes a los artículos de este número.

GUIA -PA

La Guía-Pa es una idea original de nuestro colaborador Ramiro Carballo que tiene como propósito proporcionar a las empresas informáticas una colección de currícula de programadores en formato html para su fácil y rápido acceso. Todos aquellos lectores que quieran incorporar sus datos pueden hacerlo, bien enviando su diseño web por e-mail a Ramiro Carballo (más información en la propia aplicación) o por correo en un disquete a la revista.

10 RAZONES para SUSCRIBIRSE a Programación Actual

Si la programación es tu profesión, vocación o hobby... Programación Actual es tu revista.

1. **Al suscribirte te aseguras recibir la revista en casa durante un año por el mismo precio y recibirás de regalo dos libros técnicos de programación.**

2. Todos los meses, de regalo, un CD-ROM con lo mejor en shareware de programación, versiones de evaluación de herramientas profesionales, fuentes de los artículos de la revista, demos, etc.

3. Es una publicación desarrollada por un selecto grupo de programadores con grandes conocimientos y con varios años de experiencia en el sector editorial.

4. Es la revista de vanguardia en programación, entendiendo vanguardia ir por delante en las tecnologías nuevas en programación bajo la lupa del rigor.

5. Para los que quieran dar sus primeros pasos, iniciándose en la programación, tienen su sección en **Nivel Básico**, con cursos de programación básica, C y Ensamblador.

6. La sección **Las empresas demandan** ofrecerá permanentemente cursos dedicados a aquellas áreas en las que se observa una mayor demanda de programadores por parte de las empresas, como Visual Basic, entornos Cics, MVS, Visual C y programación de bases de datos (SQL, Informix, Oracle)

7. En cuanto a **Programación para Internet** nos aseguramos ser la revista líder en España, las nuevas tendencias, las últimas normas, estandarizaciones y tecnologías futuras en la Red entrarán como exclusiva en Programación Actual.

8. Si lo que te gustan son los videojuegos, las demos y, en general, la **programación gráfica** contaremos con los mejores de la demoescene nacional, así como conocidos programadores de videojuegos colaborarán habitualmente con secciones y columnas de opinión.

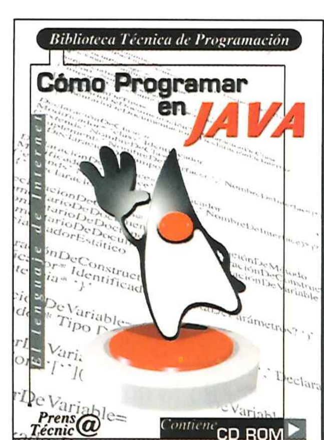
9. Finalmente, Programación Actual es una revista independiente, donde todos sus colaboradores son libres de expresar sus opiniones acerca de productos o compañías.

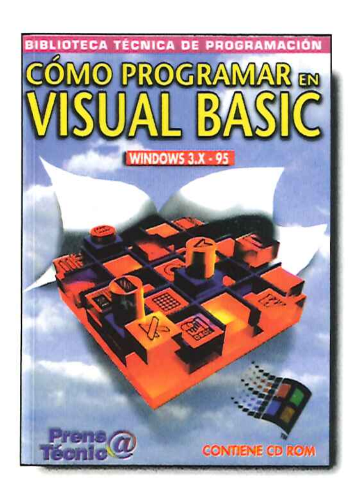
10. **Oferta especial de suscripción para estudiantes de carreras técnicas, FP II, y suscriptores de otras revistas de programación nacionales o internacionales, sólo 8.995 ptas.** (enviar fotocopia recibo suscripción o carnet estudiante según proceda).

Todos aquellos que acertéis suscribiéndoos a Programación Actual podréis elegir gratis dos super regalos de entre estos tres:

Cómo Programar en Java (Colección Biblioteca Técnica de Programación)

- Ideal para principiantes
- El lenguaje de los programadores ideal para aplicaciones de Internet
- Aplicaciones para programar en Java en el CD-ROM





Cómo Programar en Visual Basic (Colección Biblioteca Técnica de Programación)

- Permite crear aplicaciones para Windows
- Ejemplos prácticos en el CD-Rom
- Ideal para crear aplicaciones multimedia con una de las herramientas más extendidas del mundo



Cómo Trabajar Linux (Colección Biblioteca Técnica de Programación)

- Ideal para principiantes
- El sistema operativo de los expertos
- Multitud de programas y utilidades en el CD-ROM
- El proceso de instalación, TCP, Internet, X-Windows, etc.

